Cisco Unified JTAPI Developers Guide for Cisco Unified Communications Manager, Release 15

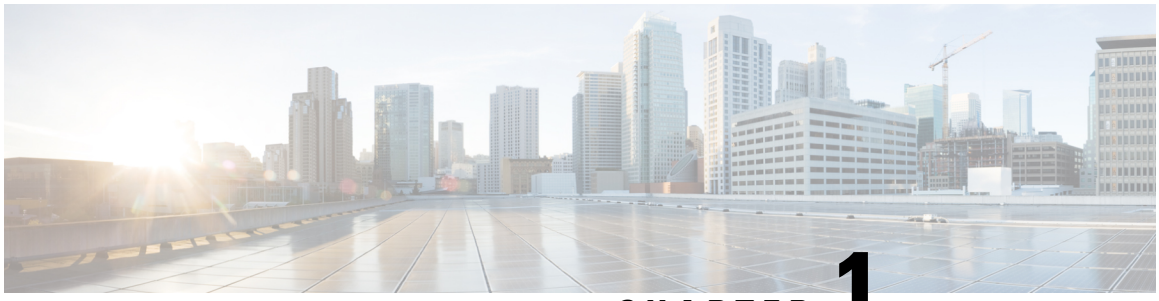
First Published: 2023-12-18

Last Modified: 2024-01-30

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1



CHAPTER 1

Overview

Cisco Unified Communications Manager (Unified CM) is the powerful call-processing component of the Cisco Unified Communications Solution. It is a scalable, distributable, and highly available enterprise IP telephony call-processing solution. Unified CM acts as the platform for collaborative communication and as such supports a wide array of features. In order to provision, invoke the features, monitor, and control such a powerful system, Unified CM supports different interface types.

This chapter gives an introduction to the different interfaces of Unified CM and describes the major concepts with which you need to be familiar before creating Java Telephony Application Programming Interface (JTAPI) applications for Cisco Unified Communications Manager systems.

For information about Cisco Unified Communications Manager features, see Features Supported by Cisco Unified JTAPI, on page 27 Also see CTI Supported Devices, on page 1655 and Cisco Unified JTAPI Operations by Release, on page 1649 for more information and CTI devices and supported features.

Cisco Unified Communications Manager Interfaces

The interface types supported by Unified CM are divided into the following types:

Provisioning Interfaces

The following are the provisioning interfaces of Unified CM:

- Administration XML
- Cisco Extension Mobility Service

Administrative XML

The Administration XML (AXL) API provides a mechanism for inserting, retrieving, updating and removing data from the Unified CM configuration database using an eXtensible Markup Language (XML) Simple Object Access Protocol (SOAP) interface. This allows a programmer to access Unified CM provisioning services using XML and exchange data in XML form, instead of using a binary library or DLL. The AXL methods, referred to as requests, are performed using a combination of HTTP and SOAP. SOAP is an XML remote procedure call protocol. Users perform requests by sending XML data to the Unified CM Publisher server. The publisher then returns the AXL response, which is also a SOAP message. For more information, See the Administrative XML Tech Center on the Cisco Developer Network http://developer.cisco.com/web/axl/home.

Cisco Extension Mobility

The Cisco Extension Mobility (Extension Mobility) service, a feature of Unified CM, allows a device, usually a Cisco Unified IP Phone, to temporarily embody a new device profile, including lines, speed dials, and services. It enables users to temporarily access their individual Cisco Unified IP Phone configuration, such as their line appearances, services, and speed dials, from other Cisco Unified IP Phones. The Extension Mobility service works by downloading a new configuration file to the phone. Unified CM dynamically generates this new configuration file based on information about the user who is logging in. You can use the XML-based Extension Mobility service API with your applications, so they can take advantage of Extension Mobility service functionality.

For more information, see the Extension Mobility API Tech Center on the Cisco Developer Network https://developer.cisco.com/site/extension-mobility/develop-and-test/documentation/latest-version/emapi-developer-guide.gsp.

Also, see *Cisco Unified Communications Manager XML Developers Guide* for relevant release of Unified CM at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Device Monitoring and Call Control Interfaces

The following are the device monitoring and call control interfaces of Unified CM:

- Cisco TAPI and Media Driver
- Cisco JTAPI
- Cisco Web Dialer

Cisco TAPI and Media Driver

Unified CM exposes sophisticated call control of IP telephony devices and soft-clients via the Computer Telephony TAPI interface. Cisco's Telephone Service Provider (TSP) and Media Driver interface enables

custom applications to monitor telephony-enabled devices and call events, establish first- and third-party call control, and interact with the media layer to terminate media, play announcements, record calls.

For more information, see the TAPI and Media Driver Tech Center on the Cisco Developer Network http://developer.cisco.com/web/tapi/home.

Also, see the *Cisco Unified TAPI Developers Guide* for Cisco Unified Communications Manager for relevant release of Unified CM at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Cisco JTAPI

For more information, see the JTAPI Tech Center on the Cisco Developer Network http://developer.cisco.com/web/jtapi/home and JTAPI Overview, on page 5.

Cisco Web Dialer

The Web Dialer, which is installed on a Unified CM server, allows Cisco Unified IP Phone users to make calls from web and desktop applications. For example, the Web Dialer uses hyperlinked telephone numbers in a company directory to allow users to make calls from a web page by clicking the telephone number of the person that they are trying to call. The two main components of Web Dialer comprise the Web Dialer Servlet and the Redirector Servlet.

For more information, see the Web Dialer Tech Center on the Cisco Developer Network https://developer.cisco.com/site/webdialer/develop-and-test/documentation/latest-version/.

For more information on Cisco Web Dialer, see the *Cisco Unified Communications Manager XML Developers Guide* for relevant release of Unified CM at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Serviceability Interfaces

The following are the serviceability interfaces of Unified CM:

- Serviceability XML
- SNMP/MIBs

Serviceability XML

A collection of services and tools designed to monitor, diagnose, and address issues specific to Unified CM serviceability XML interface:

- Provides platform, service and application performance counters to monitor the health of Unified CM hardware and software
- Provides real-time device and CTI connection status to monitor the health of phones, devices, and applications connected to Unified CM.
- Enables remote control (Start/Stop/Restart) of Unified CM services.
- Collects and packages Unified CM trace files and logs for troubleshooting and analysis.
- Provides applications with Call Detail Record files based on search criteria.

- Provides management consoles with SNMP data specific to Unified CM hardware and software.

For more information, see the Serviceability XML Tech Center on the Cisco Developer Network http://developer.cisco.com/web/sxml/home.

SNMP/MIBs

SNMP interface allows external applications to query and report various UCMgr entities. It provides information on the connectivity of the Unified Communication Manager to other devices in the network, including syslog information.

The MIBs supported by Unified CM includes:

- Cisco-CCM-MIB, CISCO-CDP-MIB, Cisco-syslog-MIB
- Standard Mibs like MIB II, SYSAPPL-MIB, HOST RESOURCES-MIB
- Vendor MIBs

For more information, see the SNMP/MIB Tech Center on the Cisco Developer Network https://developer.cisco.com/site/sxml/.

Also, see the *Cisco Unified Communications Manager XML Developers Guide* for relevant release of Unified CM at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Routing Rules Interface

Cisco Unified Communication Manager 8.0(1) and later supports the external call control (ECC) feature, which enables an adjunct route server to make call-routing decisions for Cisco Unified Communications Manager by using the Cisco Unified Routing Rules Interface. When you configure external call control, Cisco Unified Communications Manager issues a route request that contains the calling party and called party information to the adjunct route server. The adjunct route server receives the request, applies appropriate business logic, and returns a route response that instructs Cisco Unified Communications Manager on how the call should get routed, along with any additional call treatment that should get applied.

For more information, see the Routing Rules Interface Tech Center on the Cisco Developer Network https://developer.cisco.com/site/curri/develop-and-test/documentation/latest-version/.

Cisco Connection Interface

This interface has the APIs that can be invoked on a connection object. Connections retain their references to calls and addresses forever. A connection reference that is obtained from a call event can be used to obtain the connection call `(getCall())` and address `(getAddress())`.

The following are the cisco connection interfaces of Cisco Unified Communications Manager:

- Local Universal Unique Identifier of Party Associated with the Connection
- Local Universal Unique Identifier of Party Associated on the Other Side of the Call

JTAPI Overview

Cisco Unified JTAPI serves as a programming interface standard developed by Sun Microsystems for use with Java-based computer–telephony applications. Cisco JTAPI implements the Sun JTAPI 1.2 specification with additional Cisco extensions. You use Cisco JTAPI to develop applications that:

- Control and observe Cisco Unified Communications Manager phones.
- Route calls by using Computer–Telephony Integration (CTI) ports and route points (virtual devices).

Basic telephony APIs that are supported comprises conference, transfer, connect, answer, and redirect APIs.

A package of JTAPI interfaces, located in the javax.telephony.* hierarchy, defines a programming model by which Java applications interact with telephony resources. For more information about interfaces, see Cisco Unified JTAPI Classes and Interfaces, on page 1599.

This section describes the following subjects:

Cisco Unified JTAPI and Contact Centers

Cisco Unified JTAPI gets used in a contact center to monitor device status and to issue routing instructions to send calls to the right place at the right time, to start and stop recording instructions while retrieving call statistics for analysis; and to screen-pop calls into CRM applications, automated scripting, and remote call control.

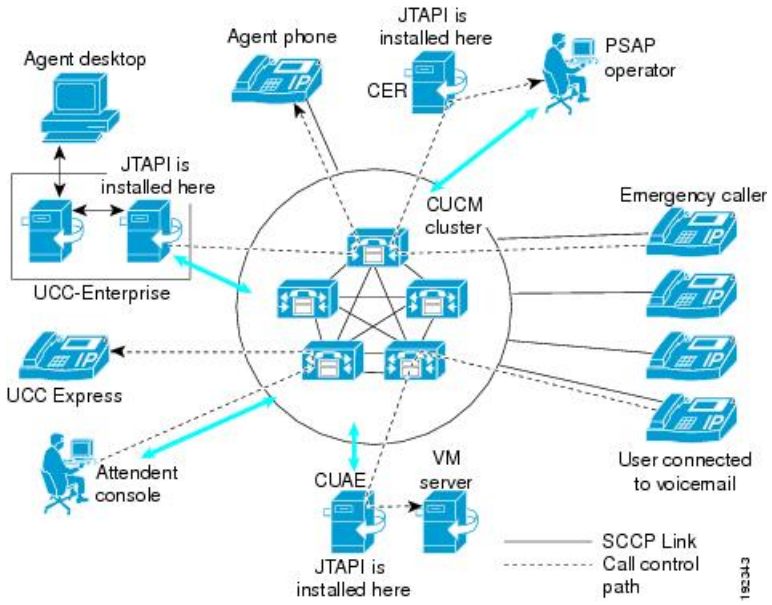
Cisco Unified JTAPI and Enterprises

Cisco Unified JTAPI, used in an enterprise environment, combines user availability, location, and preferences for a uniquely tailored environment for presence-based routing. For example, in a financial environment, market data, business logic, and call control combine in a browser-based application to enable brokers and analysts to respond to rapid changes in the global financial markets.

In a heathcare environment, call control, doctor/patient lookup, and emergency response team paging combine in a browser-based console. Further, in a hospitality environment, caller data gets linked with POS systems to automate room or restaurant reservations, dispatch taxis, and schedule wakeup calls.

The following figure shows a typical Cisco Unified Communications Manager and Cisco Unified JTAPI in an enterprise configuration.

Figure 1: Cisco Unified Communications Manager and Cisco Unified JTAPI



Cisco Unified JTAPI Applications

A Cisco Unified JTAPI application can flow as follows:

- Obtain JTAPIPeerobject instance from JTAPIPeerFactory.
- Obtain a Provider by using the getProvider() API on JTAPIPeer.
- Obtain from the Provider, the Terminal and Address for use in your application.
- Determine capabilities of relevant objects.
- Add observers for the objects that application wants to monitor/control.
- Begin application flow (for example, begin calls).

The following example shows a basic JTAPI application.

```
public void getProvider ()
{
  try
  {
    JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
    System.out.println ("Got peer "+peer);
    Provider provider  =  peer.getProvider("cti-server;login = username;passwd =
 pass");
    System.out.println ("Got provider "+provider);
    MyProviderObserver providerObserver  =  new MyProviderObserver ();
    provider.addObserver(providerObserver);
    while (outOfService )
    {
      Thread.sleep(500);
    }
    System.out.println ("Provider is now in service");
```

```
    Address[] addresses  =  provider.getAddresses();
    System.out.println ("Found "+ addresses.length +" addresses");
    for(int i = 0; i< addresses.length; i++)
    {
      System.out.println(addresses[i]);
    }
    provider.shutdown();
    catch (Exception e)
    {
    }

  }
```

Jtprefs Application

The jtapi.ini file includes parameters that are required for configuring Cisco Unified JTAPI. Cisco Unified JTAPI looks for this file in a Java classpath. The parameters get modified by using the Jtprefs application that Cisco Unified JTAPI installs. The Jtprefs application sets only the parameters that it requires. This proves beneficial because a single point of application administration exists, independent of jtapi.ini.

The jtapi.ini file contains default values, but client applications can modify values without having to specifically modify the jtapi.ini file. Different instances of client applications, however, can impose different settings for these parameters. The com.cisco.jtapi.extensions package defines the CiscoJtapiProperties interface.

Applications obtain a CiscoJtapiProperties object from the CiscoJtapiPeer and make changes to the parameters by using the accessor and mutator methods. These properties must get set and applied to all providers that are derived from a CiscoJtapiPeer prior to the first getProvider () call on that peer.

Applications that run in non GUI based platform, in which jtprefs.ini cannot be invoked, can write a jtapi.ini file and place it along with jtapi.jar.

See the following topics for more information:

- Administering User Information for JTAPI Applications, on page 236
- Fields in the jtapi.ini File, on page 236

Cisco Unified JTAPI Concepts

This section describes the following concepts:

- CiscoObjectContainer Interface, on page 8
- JtapiPeer and Provider, on page 8
- Address and Terminal Relationships, on page 10
- Connections, on page 11
- Terminal Connections, on page 11
- Terminal and Address Restrictions, on page 11
- CiscoConnectionID, on page 14

CiscoObjectContainer Interface

The CiscoObjectContainer interface allows applications to associate an application-defined object to objects that implement the interface. In Cisco Unified JTAPI, the following interfaces extend the CiscoObjectContainer interface:

- CiscoJTAPIPeer
- CiscoProvider
- CiscoCall
- CiscoAddress
- CiscoTerminal
- CiscoConnection
- CiscoTerminalConnection
- CiscoConnectionID
- CiscoCallID

JtapiPeer and Provider

The Provider object, which gets created through the implementation of the JtapiPeer object, acts as the main point of contact between applications and JTAPI implementations. The Provider object contains the entire collection of call model objects, Addresses, Terminals, and Calls, which are controllable at any time by an application.

The JTAPI Preferences (JTPREFS) application administers JtapiPeer.getServices(), which returns server names.

The Provider entails two basic processes: initialization and shutdown.

Ensure that the following information is passed in the JtapiPeer.getProvider() method for applications to obtain a CiscoProvider:

- Hostname or IP address for the Cisco Unified Communications Manager server
- Login of the user who is administered in the directory
- Password of the user that is specified
- (Optional) Application information (This parameter may comprise a string of any length.)

  Applications must include enough descriptive information, so if the appinfo were logged in and an alarm were to occur, administrators would know which application caused the alarm. Applications should not include hostname or IP address where they reside, nor the time at which they were spawned. Also, ensure that no " = " or ";" characters are included in the appinfo string because they delimit the getProvider () string. When the appinfo is not specified, you can use a generic and quasi-unique name (JTAPI[XXXX]@hostname, where XXXX represents a random, four-digit number) instead.

The parameters get passed in key value pairs that are concatenated in a string as follows:

JtapiPeer.getProvider("CTIManagerHostname;**login** = user;**passwd** = userpassword;**appinfo** = Cisco Softphone")

Initialization

The JtapiPeer.getProvider() method returns a Provider object as soon as the TCP link, the initial handshake with the Cisco Unified Communications Manager, and the device list enumeration are complete. The provider now exists in the OUT_OF_SERVICE state. Cisco Unified JTAPI applications must wait for the provider to go to the IN_SERVICE state before the controlled device list is valid. A ProvInServiceEv event gets delivered to an object that is implementing the ProviderObserver interface.

Note Implementing only the CiscoProviderObserver does not do enough; the observer must also get added to the provider with provider.addObserver(). Applications must wait for a notification that the Provider is in service.

As a part of the QoS baselining effort in JTAPI, ProviderOpenCompletedEv provides the "DSCP value for Applications" to JTAPI. JTAPI sets this DSCP value for its connection with CTI, and all JTAPI messages to CTI will have this DSCP value as long as the Provider object exists.

Shutdown

When an application calls provider.shutdown(), JTAPI loses communications permanently with the Cisco Unified Communications Manager, and a ProvShutdownEv event gets delivered to the application. The application can assume that the Provider will not come up again, and the application must handle a complete shutdown.

Provider.getTerminals()

This method returns an array of terminals that are created for the devices that are administered in the user control list in the directory. Refer to the Cisco Unified Communications Manager Administration Guide to administer the user control list.

Provider.getAddresses()

This method returns an array of addresses that are created from the lines that are assigned to the devices that are administered in the user control list in the directory.

Changes to the User Control List in the Directory

If a device is added to the user control list after the JTAPI application starts, a CiscoTermCreatedEv, and the respective CiscoAddrCreatedEv, gets generated and sent to observers that are implementing the CiscoProviderObserver. In addition, applications can monitor the current registration state of the controlled devices and dynamically track the availability of these devices. The events for an in-service Address or Terminal get delivered to observers that are implementing the CiscoAddressObserver and the CiscoTerminalObserver.

Note Implementing only the observers does not do enough; the observers must also get added by address.addObserver() and, similarly, for the terminal by the terminal.addObserver() method.

Note Before invoking the call.connect() method, add a CallObserver to the address or terminal that is originating the call; otherwise, the method returns an exception.

Address and Terminal Relationships

The Cisco Unified Communications system architecture includes three fundamental types of endpoints:

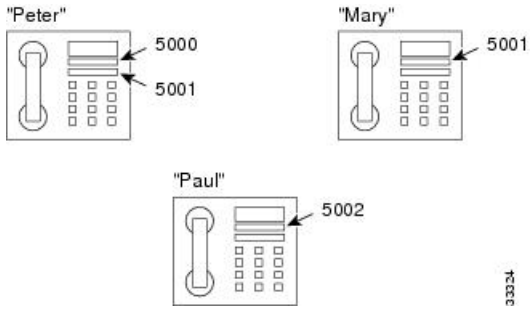
- Phones
- Virtual devices (media termination points and route points)
- Gateways

Of these endpoints, only phones and media termination points get used by using the Cisco Unified JTAPI implementation.

Cisco Unified Communications Manager allows users to configure phones to have one or more lines, dialable numbers, which multiple phones may share simultaneously, or lines can be configured for exclusive use by only one phone at a time. Each line on a phone can terminate two calls simultaneously, one of which must be on hold.

This operation acts in a similar way to the operation of the "call waiting" feature on home phones. Figure 2: Phone Diagram, on page 10 shows two configurations: Peter and Mary share one phone line, 5001, while Paul has his own phone line, 5002.

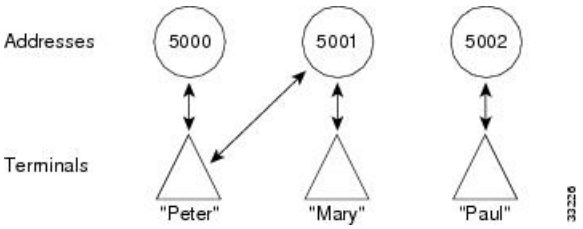
Figure 2: Phone Diagram



A unique name identifies all types of Cisco Unified Communications Manager endpoints. The phone Media Access Control (MAC) address (such as, "SEP0010EB1014") identifies it, and the system administrator can assign any name to a media termination point, so long as its name is unique.

For each endpoint that a provider controls, the Cisco Unified JTAPI implementation uses the administrator-assigned name to construct a corresponding terminal object. Terminal objects in turn have one or more address objects, each of which corresponds to a line on the endpoint. Figure1-2 "Address and Terminal Relationship" shows a graphical representation of the relationship between addresses and terminals.

Figure 3: Address and Terminal Relationship



If two or more endpoints share a line (DN), the corresponding address object relates to more than one terminal object.

Unobserved Addresses and Terminals

Cisco Unified JTAPI perceives calls only when a CallObserver attaches to the terminals and addresses of the provider. This means that methods such as Provider.getCalls() or Address.getConnections() will return null, even when calls exist at the address, unless a CallObserver attaches to the address. The system also requires adding a CallObserver to the address or terminal that is originating a call via the Call.connect() method.

Connections

Connections retain their references to calls and addresses forever. So, you can always use a connection reference that is obtained from a call event to obtain the connection call (getCall()) and address (getAddress()).

Terminal Connections

Terminal connections always retain their references to terminals and connections. So, you can always use a terminal connection reference that is obtained from a call event to obtain the terminal connection terminal (getTerminal()) and connection (getConnection()).

Terminal and Address Restrictions

Terminal and address restrictions prohibit applications from controlling and monitoring a certain set of terminals and addresses when the administrator configures them as restricted in Cisco Unified Communications Manager Administration.

The administrator can configure a particular line on a device (address on a particular terminal) as restricted. If a terminal is added into the restricted list in Cisco Unified Communications Manager Administration, all addresses on that terminal also get marked as restricted in JTAPI. If an application comes up after the configuration completes, it can perceive whether a particular terminal or address is restricted from checking the interface CiscoTerminal.isRestricted() and CiscoAddress.isRestricted(Terminal). For shared lines, applications can query the interface CiscoAddress.getRestrictedAddrTerminals(), which indicates whether an address is restricted on any terminals.

If a line (address on a terminal) is added into the restricted list after an application comes up, the applications will perceive CiscoAddrRestrictedEv. If the address has any observers, applications will recognize CiscoAddrOutOfService. When a line is removed from the restricted list, applications will perceive CiscoAddrActivatedEv. If an address has any observers, applications see CiscoAddrInServiceEv. Ifan application tries to add observers on an address after it is restricted, a PlatformException gets thrown. However, if any observers are added before the address is restricted, they will remain as is, but applications cannot get any events on these observers unless the address is removed from the restricted list. Applications can also choose to remove observers from an address.

If a device (terminal) is added to the restricted list after an application comes up, the application will see CiscoTermRestrictedEv. If the terminal has any observers, the application will see CiscoTermOutOfService. If a terminal is added to the restricted list, JTAPI also restricts all addresses that belong to that terminal and applications will perceive CiscoAddrRestrictedEv. If a terminal is removed from the restricted list, applications will perceive CiscoTermActivatedEv and CiscoAddrActivatedEv for the corresponding addresses. If an application tries to add observers on a terminal after it is added to the restricted list, a PlatformException is thrown. However, if any observers are added before the terminal is restricted, they will remain as is, but applications cannot get any events on these observers unless the terminal is removed from the restricted list

If a shared line is added to the restricted list after an application comes up, the application will perceive CiscoAddrRestrictedOnTerminalEv. If any address observers exist on the address, the application will recognize CiscoAddrOutOfServiceEv for that terminal. If all shared lines are added to the restricted list, when the last one is added, applications will perceive CiscoAddrRestrictedEv. If a shared line is removed from the restricted list after the application comes up, applications will perceive CiscoAddrActivatedOnTerminalEv. If any observers exist on the address, the application will perceive CiscoAddrInServiceEv for that terminal. Ifall shared lines in the control list are removed from the restricted list, applications will recognize CiscoAddrActivatedEv when the last one is removed, and all addresses on terminals will receive InService events.

If all shared lines in the control list are marked as restricted, and an application tries to add observers, a platform exception gets thrown. If a few shared lines are in the restricted list, while others are not, when an application adds an observer on the address only nonrestricted lines will go in service.

If any active calls are present when an address or terminal is added to the restricted list and reset, applications will recognize connection and TerminalConnections get disconnected.

If no addresses or terminals are added to the restricted list, this feature remains backward compatible with earlier versions of JTAPI: no new events get delivered to applications.

The following sections describe the interface changes for address and terminal restrictions.

CiscoTerminal

boolean	`isRestricted()` Indicates whether a terminal is restricted. If the terminal is restricted, all associated addresses on this terminal also get restricted. Returns true if the terminal is restricted; returns false if it is not restricted.

CiscoAddress

javax.telephony.Terminal[]	`getRestrictedAddrTerminals()` Returns an array of terminals on which this address is restricted. If none are restricted, this method returns null. In shared lines, a few lines on terminals may get restricted. This method returns all the terminals on which this particular address is restricted. Applications cannot perceive any call events for restricted lines. If a restricted line is involved in a call with any other control device, an external connection gets created for the restricted line.
boolean	`isRestricted(javax.telephony.Terminal terminal)` Returns true if any address on this terminal is restricted.Returns false if no addresses on this terminal are restricted.

```
public interface CiscoRestrictedEv extends CiscoProvEv {
  public static final int ID  =  com.cisco.jtapi.CiscoEventID.CiscoRestrictedEv;

  /**
   * The following define the cause codes for restricted events
   */

  public final static int CAUSE_USER_RESTRICTED  =  1;

  public final static int CAUSE_UNSUPPORTED_PROTOCOL  =  2;

}
```

This represents the base class for restricted events and defines the cause codes for all restricted events. CAUSE_USER_RESTRICTED indicates the terminal or address is marked as restricted. CAUSE_UNSUPPORTED_PROTOCOL indicates that the device in the control list is using a protocol that Cisco Unified JTAPI does not support. Existing Cisco Unified IP 7960 and 7940 phones that are running SIP fall in this category.

CiscoAddrRestrictedEv

Public interface **CiscoAddrRestrictedEv** extends CiscoRestrictedEv. Applications will recognize this event when a line or an associated device is designated as restricted from Cisco Unified Communications Manager Administration. For restricted lines, the address will go out of service and will not come back in service until it is activated again. If an address is restricted, addCallObserver and addObserver will throw an exception. For shared lines, if a few shared lines are restricted, and others are not, no exception gets thrown, but restricted shared lines will not receive any events. If all shared lines are restricted, an exception gets thrown when observers are added. If an address is restricted after observers are added, applications will perceive CiscoAddrOutOfServiceEv, and when the address is activated, the address will go in service.

CiscoAddrActivatedEv

Public interface **CiscoAddrActivatedEv** extends CiscoProvEv. Applications will perceive this event whenever a line or an associated device is in the control list and is removed from the restricted list in the Cisco Unified Communications Manager Administration. If any observers exist on the address, applications will perceive CiscoAddrInServiceEv. If no observers exist, applications can try to add observers, and the address will go in service.

CiscoAddrRestrictedOnTerminalEv

Public interface **CiscoAddrRestrictedOnTerminalEv** extends CiscoRestrictedEv. If a user has a shared address in the control list, and if one of the lines is added into the restricted list, this event will get sent. Interface getTerminal() returns the terminal on which the address is restricted. Interface getAddress() returns the address that is restricted.

javax.telephony.Address	`getAddress()`
javax.telephony.Terminal	`getTerminal()`

CiscoAddrActivatedOnTerminal

Public interface **CiscoAddrActivatedOnTerminalEv** extends CiscoProvEv. When a shared line or a device that has a shared line is removed from the restricted list, this event will get sent. The interface getTerminal() returns the terminal that is being added to the address. The interface getAddress() returns the address on which the new terminal is added.

javax.telephony.Address	`getAddress()`
javax.telephony.Terminal	`getTerminal()`

CiscoTermRestrictedEv

Public interface **CiscoTermRestrictedEv** extends CiscoRestrictedEv. Applications will perceive this event when a device is added into restricted list from Cisco Unified Communications Manager Administration after the application launches. Applications cannot perceive events for restricted terminals or addresses on those terminals. If a terminal is restricted when it is in InService state, applications will get this event, and terminal and corresponding addresses will move to the out-of-service state.

CiscoTermActivatedEv

Public interface **CiscoTermActivatedEv** extends CiscoRestrictedEv.

javax.telephony.Terminal	`getTerminal()` Returns the terminal that is activated and is removed from the restricted list.

CiscoOutOfServiceEv

static int	`CAUSE_DEVICE_RESTRICTED` Indicates whether an event is sent because a device is restricted.
static int	`CAUSE_LINE_RESTRICTED` Indicates whether an event is sent because a line is restricted.

CiscoCallEv

static int	`CAUSE_DEVICE_RESTRICTED` Indicates whether an event is sent because a device is restricted.
static int	`CAUSE_LINE_RESTRICTED` Indicates whether an event is sent because a line is restricted.

CiscoConnectionID

The CiscoConnectionID object represents a unique object that is associated with each connection in Cisco Unified JTAPI. Applications may use the object itself or the integer representation of the object.

Threaded Callbacks

The Cisco Unified JTAPI implementation design allows applications to invoke blocking JTAPI methods such as Call.connect() and TerminalConnection.answer() from within their observer callbacks. This means that

applications do not get subjected to the restrictions that are imposed by the JTAPI 1.2 specification, which cautions applications against using JTAPI methods from within observer callbacks.

CiscoSynchronousObserver Interface

The Cisco Unified JTAPI implementation allows applications to invoke blocking JTAPI methods, such as Call.connect() and TerminalConnection.answer(), from within observer callbacks. This means that applications do not get subjected to the restrictions that the JTAPI 1.2 specification imposes, which cautions against using JTAPI methods from within observer callbacks. Applications can selectively disable the queuing logic of the Cisco Unified JTAPI implementation by implementing the CiscoSynchronousObserver interface on their observer objects.

This asynchronous behavior does not adversely affect many applications. Applications that would benefit from a coherent call model during observer callbacks can selectively disable the queuing logic of the Cisco Unified JTAPI implementation. By implementing the CiscoSynchronousObserver interface on its observer objects, an application declares deliver synchronous events to its observers. Events that are delivered to synchronous observers will match the states of the call model objects that are queried from within the observer callback.

Note Objects that implement the CiscoSynchronousObserver interface may not invoke blocking JTAPI methods from within their event callbacks. The consequences of doing so are unpredictable, and may include deadlocking the JTAPI implementation. On the other hand, you may safely use the access or methods of any JTAPI object, such as Call.getState() or Connection.getState(). Applications should avoid calling any interface that returns an array such as Terminal.getAddresses() in synchronous callbacks.

Querying Dynamic Objects

Beware of querying dynamic objects such as call objects. By the time you get an event, the object (such as, call) may exist in a different state than the state that is indicated. For example, by the time you get a CiscoTransferStartEV, the transferred call may have removed all its internal connections.

callChangeEvent()

When the callChangedEvent() method is called, the validity remains guaranteed for any references that are contained in the event. For example, if the event contains a getConnection() method, the application can call this method and get a valid connection reference. Likewise, a getCallingAddress() method guarantees to return a valid Address object.

CiscoConsultCall

For the CiscoConsultCall interface, a reference to a consulting terminal connection gets retained forever. For example, when a CiscoConsultCallActive event is processed, getConsultingTerminalConnection() guarantees to return a valid terminal connection reference. Further, the terminal connection guarantees to provide access to the consulting connection and thus the consulting call.

CiscoTransferStartEv

For the CiscoTransferStartEv, the references to the transferred call, transfer controller, and final call in the event become valid when callChangedEvent() is called. However, getConnections() may or may not return the connections on these calls.

Alarm Services

Part of the general serviceability framework for Cisco Unified Communications applications includes support for sending alarms to a service. The com.cisco.services.alarm package defines the alarm components.

An alarm interface and framework support the sending of alarm notifications in XML over TCP to an Alarm Service that is available on the network in a Cisco Unified JTAPI application. The alarm package includes the following features:

- XML definition of alarms, resolved by a catalog in the alarm service
- A bounded rollover queue to buffer alarms at the sender
- Alarm sending on a separate thread to avoid blocking at the sending application
- A TCP-based reconnection scheme to the alarm service

The overall framework of the Cisco Unified JTAPI alarm system includes similarities to the existing JTAPI tracing package. Applications must instantiate an AlarmManager for a particular facility code from which alarm objects can be created. Part of the implementation includes DefaultAlarm and DefaultAlarmWriter implementation classes.

Software Requirements

The following table lists the software requirements for JTAPI applications, JTPREFS, and sample code.

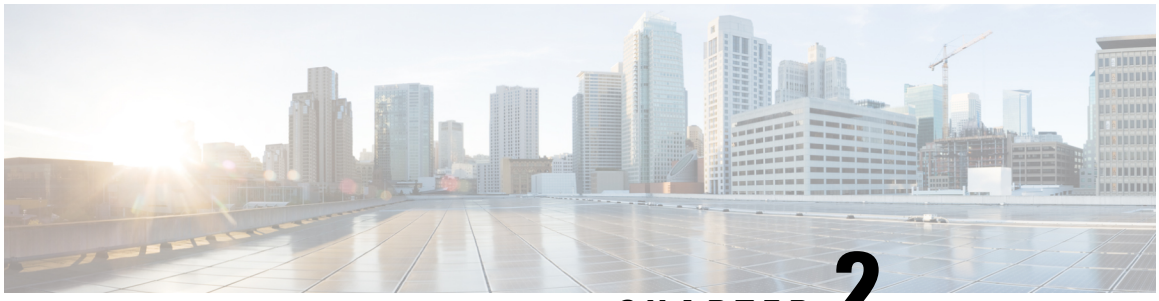
Application	Required Software
JTAPI applications	Any JDK 1.6 compliant Java environment
JTPREFS	Any JDK 1.6 compliant environment.
JTPREFS	Any JDK 1.6 compliant Java environment

Development Guidelines

Cisco maintains a policy of interface backward compatibility for at least one previous major release of Cisco Unified Communications Manager (Cisco Unified CM). Cisco still requires Cisco Technology Developer Program member applications to be retested and updated as necessary to maintain compatibility with each new major release of Cisco Unified CM.

The following practices are recommended to all developers, including those in the Cisco Technology Developer Program, to reduce the number and extent of any updates that may be necessary:

- The order of events and/or messages may change. Developers should not depend on the order of events or messages. For example, where a feature invocation involves two or more independent transactions, the events or messages may be interleaved. Events related to the second transaction may precede messages related to the first. Additionally, events or messages can be delayed due to situations beyond control of the interface (for example, network or transport failures). Applications should be able to recover from out of order events or messages, even when the order is required for protocol operation.
- The order of elements within the interface event or message may change, within the constraints of the protocol specification. Developers must avoid unnecessary dependence on the order of elements to interpret information.
- New interface events, methods, responses, headers, parameters, attributes, other elements, or new values of existing elements, may be introduced. Developers must disregard or provide generic treatments where necessary for any unknown elements or unknown values of known elements encountered.
- Previous interface events, methods, responses, headers, parameters, attributes, and other elements, will remain, and will maintain their previous meaning and behavior to the extent possible and consistent with the need to correct defects.
- Applications must not be dependent on interface behavior resulting from defects (behavior not consistent with published interface specifications) since the behavior can change when defect is fixed.
- Use of deprecated methods, handlers, events, responses, headers, parameters, attributes, or other elements must be removed from applications as soon as possible to avoid issues when those deprecated items are removed from Cisco Unified CM.
- Application Developers must be aware that not all new features and new supported devices (for example, phones) will be forward compatible. New features and devices may require application modifications to be compatible and/or to make use of the new features/devices.



CHAPTER 2

New and Changed Information

This chapter describes new and changed JTAPI information for this release of Cisco Unified Communications Manager and features supported in the previous releases.

For more information, go to the Programming Guides website at http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Cisco Unified Communications Manager Release 15

There are no new or changed JTAPI specifications for Unified Communications Manager Release 15.

Cisco Unified Communications Manager Release 14SU2

This section contains information about the new and changed features for Unified Communications Manager Release 14SU2:

FIPS Compliance, on page 93

Cisco Unified Communications Manager Release 12.5(1)

This section contains information about the new and changed features for Cisco Unified Communications Manager, Release 12.5(1):

- Call Recording for SIP or TLS Authenticated Calls, on page 41
- Multi-fork Recording using CUBE Media Proxy Server, on page 122
- Linux and Windows installation procedure is updated in *Installing the Cisco Unified JTAPI Software* section.

Cisco Unified Communications Manager, Release 11.5(1)

This section contains information about the new and changed features for Cisco Unified Communications Manager, Release 11.5(1):

- Hunt Log Status, on page 102
- End to End Session ID for Calls, on page 92
- Redirect to Device, on page 146
- SHA-512 Support for Digital Signatures, on page 190

Starting from **Release 11.5(1)SU9** and any subsequent SU or ES releases in this release train, the Cisco JTAPI Plugin follows installer less approach. You must have JRE installed on the system before the installation. The installation runs in the command prompt and does not have a GUI. Also, the same will not be listed in the software installed list of Windows Control Panel.

Note Cisco Spark Device has been added as a new device type for this release of Unified Communications Manager and may appear in the user's control list. However, Cisco Spark Device is not a supported device for this release of Cisco Unified JTAPI.

Cisco Unified Communications Manager, Release 11.0(1)

This section contains information about the new and changed features for Cisco Unified Communications Manager, Release 11.0(1).

- Default CTI IP Addressing for Devices, on page 73
- Ringback on SIP 183 for Transferred Calls, on page 150

Cisco Unified Communications Manager Release 10.5(2)

This section contains the new and changed features for Cisco Unified Communications Manager release 10.5(2):

- AES 256 Algorithm IDs, on page 32

Cisco Unified Communications Manager Release 10.0(1)

This section describes the new and changed features in Cisco Unified Communications Manager Release 10.0(1):

- CTI RD Call Forward, on page 70
- CTI Video Support, on page 71
- Encryption Enhancement, on page 87
- Mobility Interaction Support, on page 70
- NuRD (Number Matching for Remote Destination) Support, on page 69
- Play Announcement, on page 68
- Persistent Connection, on page 132
- SSO Cookie, on page 168
- Recording, on page 141
- Verify Remote Destination Support, on page 69
- Video Capabilities and Multi-Media Information, on page 205
- Video On Hold Support, on page 209

Cisco Unified Communications Manager Release 9.0(1)

This section describes the new and changed features in Cisco Unified Communications Manager release 9.0(1):

- Cius Persistency, on page 57
- CTI Remote Device for JTAPI, on page 67
- E911 Teleworker, on page 86
- Hunt List Connected Number, on page 102
- Native Queuing, on page 122
- URI Dialing, on page 204

Cisco Unified Communications Manager Release 8.6(1)

This section describes the new and changed features in Cisco Unified Communications Manager release 8.6(1):

- Account Lockout, on page 31
- EnergyWise Deep Sleep Mode, on page 88
- FIPS Compliance, on page 93
- Password Expiry, on page 132
- New JTAPI x64 client for 64-bit operating systems.

Cisco Unified Communications Manager Release 8.5(1)

This section describes the new and changed features in Cisco Unified Communications Manager release 8.5(1):

- Agent Greeting, on page 31
- API for Exposing Built-In-Bridge Status, on page 33
- Play Zip Tone, on page 134
- Single Sign-On, on page 167
- Support for VMware, on page 183

Cisco Unified Communications Manager Release 8.0(1)

This section describes the new and changed features in Cisco Unified Communications Manager release 8.0(1):

- Call Control Discovery, on page 39
- Call Pickup, on page 40
- CallFwdAll Key Press Notification, on page 44
- End to End Call Tracing, on page 87
- Extension Mobility Cross Cluster, on page 90
- External Call Control, on page 91
- Hunt List, on page 101
- iSac Codec, on page 107
- Secured Monitoring and Recording, on page 159
- Support for Cisco Unified IP Phone 6901, on page 180
- Support for 100+ Directory Numbers, on page 182
- Support for VMware, on page 183
- Verification Involving PSTN Reachability, on page 205

Cisco Unified Communications Manager Release 7.1(3)

This section describes the new and changed features in Cisco Unified Communications Manager release 7.1(3):

• Terminal and Address Capability Settings, on page 185.

Cisco Unified Communications Manager Release 7.1(2)

This section describes the new and changed features in Cisco Unified Communications Manager release 7.1(2):

• Component Updater, on page 60
• Direct Transfer Across Lines, on page 75
• Drop Any Party, on page 83
• IPv6 Support, on page 106
• Join Across Lines or Connected Conference Across Lines, on page 109
• Logical Partitioning, on page 116
• Message Waiting Indicator Enhancement, on page 119
• Park Monitoring and Assisted DPark Support, on page 126
• Swap or Cancel and Transfer or Conference Behavior, on page 184

Cisco Unified Communications Manager Release 7.0(1)

This section describes the new and changed features in Cisco Unified Communications Manager from release 6.1 to release 7.0(1) and Cisco Unified JTAPI enhancements. It has the following sections:

• Call Pickup, on page 40
• Calling Party Normalization, on page 44
• Click to Conference, on page 58
• Do Not Disturb-Reject, on page 82
• Extension Mobility Username Login, on page 91
• Java Socket Connect Timeout, on page 107
• Join Across Lines with Conference Enhancements (SCCP and SIP), on page 113
• Locale Infrastructure Development, on page 115
• selectRoute() with Calling Search Space and Feature Priority, on page 161

Note Cisco Unified Communications Manager release 7.0(1) does not support the following IPv6 related methods:

canSupportIPv6()

setProviderOpenRetryAttempts (int retryAttempts)

getProviderOpenRetryAttempts()

getIPAddressingMode() (*available on CiscoMediaTerminal and CiscoRouteTerminal interfaces*)

register(java.net.InetAddress address, int port, CiscoMediaCapability [] capabilities, int[] algorithmIDs, java.net.InetAddress address_v6, int activeAddressingMode)

register(CiscoMediaCapability [] capabilities, int[] int registration Type, int[] algorithmIDs, int activeAddressingMode)

getTerminals() (*available on new interface CiscoProviderTermCapabilityChangedEv*)

getAddressingModeForMedia()

getCallingPartyIpAddr_v6() (*available on CiscoCallCtlConnOfferedEv and CiscoRouteEvent interfaces*)

CTIERR_IPADDRMODEMISMATCH

CTIERR_DYNREG_IPADDRMODE_MISMATCH

hasIPv6CapabilityChanged()

CiscoTerminal.IP_ADDRESSING_MODE_IPv4

CiscoTerminal.IP_ADDRESSING_MODE_IPv6

CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6

CiscoTerminal.IP_ADDRESSING_MODE_Unknown

CiscoTermRegistrationFailedEv.IP_ADDRESSING_MODE_MISMATCH

Note For the features, Join Across Lines, Do Not Disturb-Reject, and Calling Party Normalization, each Cisco JTAPI must be upgraded to a version that supports these features. Additionally, if you are upgrading from release 5.1 and you use Join Across Lines, the Conference Chaining feature must not be enabled or used until all applications are either upgraded to a version compatible with the new unified CM version. Also, you should verify that the applications are not impacted by the Conference Chaining feature.

Cisco Unified Communications Manager Release 6.1

This section describes the new and changed features in Cisco Unified Communications Manager from release 6.0 to release 6.1 and Cisco Unified JTAPI enhancements. It has the following sections:

- Certificate Download API Enhancement, on page 45
- Intercom Support for Extension Mobility, on page 105
- Join Across Lines, on page 108

Cisco Unified Communications Manager Release 6.0

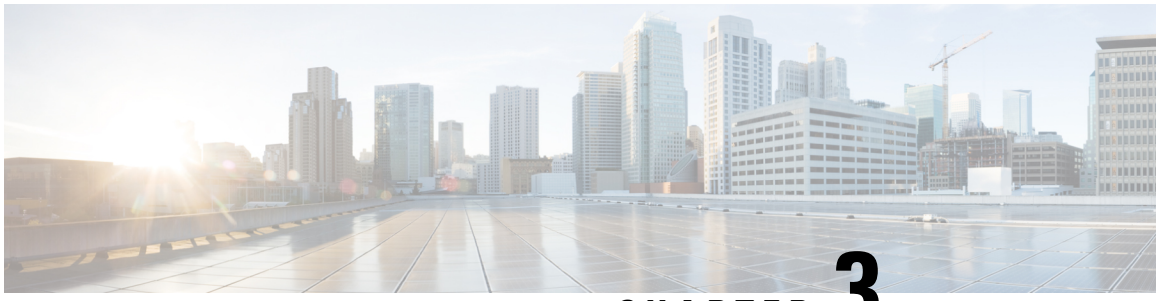
This section describes the new and changed features in Cisco Unified Communications Manager, release 6.0 and Cisco Unified JTAPI enhancements. It has the following sections:

Cisco Unified Communications Manager Release 5.1

This section describes the new and changed features in Cisco Unified Communications Manager, from release 5.0 to release 5.1 and Cisco Unified JTAPI enhancements. It has the following sections:

Cisco Unified Communications Manager Release 5.0

This section describes the new and changed features in Unified Communications Manager, from release 4.x to release 5.0 and Cisco Unified JTAPI enhancements. It has the following:



CHAPTER 3

Features Supported by Cisco Unified JTAPI

This chapter describes features supported by the Cisco Unified JTAPI specification.

Account Lockout

The administrator can use the CUCM Admin Panel to configure options for the account lockout.

To configure account lockout options, an administrator can perform either of the following:

1. Click the Locked by Administrator checkbox in the user credential page.
2. Set the number of login attempts, which signifies the number of failed logins due to invalid credentials.
3. Set the maximum idle time (in days) and if the user does not login for that many days, the account is locked.

In case of account lockout, JTAPI delivers detailed exceptions without any warning messages. JTAPI does not allow applications to modify any of these values, it only reports the information.

Interface Changes

CiscoJtapiExceptions, on page 53

Message Sequences

There are no message sequences.

Backward Compatibility

This feature is backward compatible.

Agent Greeting

The Agent Greeting feature enables the JTAPI application to instruct the Cisco Unified Communications Manager to automatically play a pre-recorded announcement following a successful media connection to the agent device. The greeting helps to keep the agent sounding fresh as they do not have to repeat common phrases on each call. Agent Greeting is audible for the agent and the customer.

Agent Greeting can be initiated from any phone with a Built-in-Bridge (BIB). A call is initiated from the BIB to the DN specified in the request. Applications are responsible for answering this call and playing the media.

There are two types of calls:

- A basic call between the customer and agent.
- A secondary call, known as the Interactive Voice Response (IVR) call, which is created between an IVR device and the BIB of the agent phone.

The application invokes the new Agent Greeting API on a call, which creates an IVR call. The application then answers the call, and is responsible to play a recorded message.

The connection is not created for the agent on the IVR call, and as a result, the applications see the secondary call only. The IVR call has only one connection to play the IVR message.

Regardless of whether or not the application observes the IVR device, the Agent Greeting media plays. Observers on the agent receive an event to start the media. When the media finishes, the application must

disconnect the IVR or CTI port that streams the media. When the second call is disconnected, an event is sent to observers on the agent and receives an event to end the media.

This feature is available only on phones that have BIBs. The majority of Cisco Unified IP Phones have BIBs, but the feature may not be available in various older or lower-end phone models. Administrators must enable the BIB for the device and configure it using the Cisco Unified Communications Manager Admin panel.

Whenever a request to addMediaStream is made, JTAPI blocks the request until the IVR device answers the call or CTI responds with a timeout error. Due to this, the JTAPI thread that invoked the addMediaStreamRequest cannot answer its own call, because it is blocked waiting for the request to finish.

Applications intending to use this feature must ensure that one of the following is applicable:

- The IVR DN is configured to auto-answer incoming calls
- A separate JTAPI thread or application is set up to answer on the IVR DN

  Interface changes

See CiscoTerminalConnection, on page 630, CiscoFeatureReason, on page 402, CiscoJtapiException, on page 410, CiscoMediaStreamStartedEv, on page 425, CiscoMediaStreamEndedEv, on page 426

Message Sequences

See Agent Greeting, on page 756

Backward Compatibility

This feature is backward compatible.

- This is a new feature and has no impact on existing features.
- There are two new events for this feature, but they are only generated if the application observes the addresses in which the feature is invoked.
- The odd call model for the IVR call, with only one connection, can have implications for applications that look at the number of connections for any of their logic.
- Feature interaction is not supported on IVR calls.

  For example, invoking features such as redirect and creating a conference from the IVR call are not supported.
- The IVR call is intended to stream media. Applications invoke features on the IVR call at their own risk and there are no event flows or call diagrams for any feature interaction on the IVR calls.

AES 256 Algorithm IDs

From release 10.5(2) Cisco Unified Communications Manager now supports the following encryption algorithm IDs:

- CiscoMediaEncryptionAlgorithmType
- CiscoMediaEncryptionAlgorithmType.AES_128_COUNTER_80
- CiscoMediaEncryptionAlgorithmType.F8_128_COUNTER_32

- CiscoMediaEncryptionAlgorithmType.F8_128_COUNTER_80
- CiscoMediaEncryptionAlgorithmType.AEAD_128_COUNTER
- CiscoMediaEncryptionAlgorithmType.AEAD_256_COUNTER

The CiscoMediaEncryptionAlgorithmType.AEAD_128_COUNTER and CiscoMediaEncryptionAlgorithmType.AEAD_256_COUNTER will be negotiated only for a secure call between two SIP endpoints.

CTI ports can register with any of the above algorithms, but will negotiate on AES_128_COUNTER_80 for secure calls.

Note From Release 12.5(1)SU5 onwards, CTI ports can register with any of the above algorithms for secure calls. For more information, see "Stronger Cipher Suites on CTI Ports" section in Security Guide for Cisco Unified Communications Manager.

Alternate Script Support

Certain IP phone types support an alternate language script other than the default script that corresponds to the phone-configurable locale. For example, the Japanese phone locale has two written scripts. Some phone types support only the default Katakana script, while other phones types support both the default script and the alternate Kanji script. Because applications can send text information to the phone for display purposes, they need to know what alternate script a phone supports, if any.

The new getAltScript() method provides alternate script information for an observed device. Currently there is only one known alternate script: Kanji for the Japanese locale.

JTAPI provides a new method for CiscoTerminal to provide alternate script information.

java.lang.String	`getAltScript()` Only one alternate script, Kanji for the Japanese locale, is currently supported. An empty string return value indicates there is no alternate script configured or the terminal does not support an alternate script.

Backward Compatibility

The alternate script feature does not impact JTAPI backward compatibility.

API for Exposing Built-In-Bridge Status

JTAPI exposes the API, CiscoTerminal.isBuiltInBridgeEnabled() to let applications know if the BIB capability is enabled on the terminal or not. Accordingly, the return value is true or false.

This API throws MethodNotSupportedException if it is invoked on a CiscoMediaTerminal or a CiscoRouteTerminal as these devices do not support a BIB.

This API throws InvalidStateException if invoked on a terminal that is not registered with the Cisco Unified Communications Manager.

Interface Changes

See CiscoTerminal, on page 611

Message Sequences

See API for Exposing Built-in-Bridge Status, on page 760

Backward Compatibility

This change is backward compatible and does not affect the existing applications.

Arabic and Hebrew Language Support

This version of the Cisco Unified JTAPI supports the Arabic and Hebrew languages, which users may select during installation and in the Cisco Unified JTAPI Preferences user interface.

Backward Compatibility

This feature is backward compatible.

Auto Updater for Linux

In order to support this feature for Linux based JTAPI client machines, auto updater feature has the following changes in its interface. The interface required that applications provide component name, provider IP address, user name and password. Applications do not need to specify an URL for downloading the component. This is done to avoid the issue with updater application in case URL changes between various releases of Cisco Unified Communications Manager Administration.

A new API called “Replace()” is part of the component interface. This facilitates replacing of old component with a newly downloaded component. The following section defines the operation of updater after the new interface changes. The new updater will:

- Use the same API signature as the old one.
- Create a file newjtapi.jar in the current folder of application which is the new version of the jar file.
- Copy the current jtapi.jar to a file by name component.temp in the classpath specified.
- Replace the current jar file with the new jar file. At the end of this operation, the current jar file becomes the component.temp and new jar file becomes jtapi.jar. Applications can still use old component interface which take URL either by specifying the URL themselves or by querying the URL through the new interface provided on CiscoProvider. The API required to get the URL information is present in the Interface summary for this feature. This operation is supported for both Unix and Windows.

  Backward compatibility

This feature is not backward compatible.

AutoAccept Support for CTI Ports and Route Points

This feature provides applications with the ability to enable or disable AutoAccept for the addresses on CTIPorts and Route Points. When AutoAccept status changes for the address, Cisco Unified JTAPI provides the event to inform the application for changes.

Note The maximum number of lines that are supported for route points equals 34.

The new interface setAutoAcceptStatus(), provided on the CiscoAddress object, allows the capability to set AutoAccept to ON or OFF. Interface getAutoAcceptStatus(), also provided on the CiscoAddress object, allows applications to query the current status of AutoAccept on the address.

When AutoAccept status changes for the address, applications get CiscoAddrAutoAcceptStatusChangedEv on AddressObservers. This event includes the interface getTerminal(), which returns the terminal on which the AutoAccept status gets changed, and the interface getAutoAcceptStatus(), which returns integers that specify whether AutoAccept is ON or OFF. If an address observer is not added, the event does not get provided.

The following interfaces support AutoAccept on CTIPort and RoutePoint:

Cisco Address

- init

  ```
  init getAutoAcceptStatus (javax.telephony.Terminal terminal)
  ```

  Ciscoaddress.getAutoAccept(Terminal iterminal) returns an AutoAccept status of address on terminal.

- void

  ```
  setAutoAcceptStatus (int autoAcceptStatus,  javax.telephony.Terminal terminal)
  ```

  This allows an application to enable AutoAccept for addresses on the CiscoMediaTerminal and or the CiscoRouteTerminal.

CiscoAddrAutoAcceptStatusChangedEv

CiscoAddrAutoAcceptStatusChangedEv

Public interface: **`CiscoAddrAutoAcceptStatusChangedEv`**

Extends com.cisco.jtapi.exension.CiscoAddrEv

The CiscoAddrAutoAcceptStatusChangedEv event gets sent to applications whenever AutoAccept status for the address on the terminal gets changed. If an address has multiple terminals, this event gets sent for the address AutoAccept status on each individual terminal.

This event provides the following interface:

- init

  ```
  getAutoAcceptStatus ()
  ```

  CiscoAddrAutoAcceptStatusChangedEv.getAutoAcceptStatus returns the following value of AutoAccept status of address on terminal CiscoAddress.AUTOACCEPT_OFF CiscoAddress.AUTOACCEPT_ON.

- com.cisco.jtapi.extensions.CiscoTerminal

  `getTerminal ()`

  Returns the terminal at which this address AutoAccept status gets changed.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for AutoAccept on CTIPort and RoutePoint, see Message Sequence Charts, on page 755

Autoupdate of API

Be aware that when the Cisco Unified Communications Manager is upgraded to a higher version, the APIs may or may not be compatible with the new Cisco Unified Communications Manager version. Ensure that the APIs are upgraded to a compatible version, so the applications work as expected. Because the APIs are installed locally on the client server, the upgrade must take place on multiple machines. In the case of fewer client applications, you can easily do this by connecting to the Cisco Unified Communications Manager Administration and downloading and installing the Cisco Unified Communications Manager compatible plug-in.

For multiple client applications, this feature provides a facility by which an application at startup can identify itself to a web server via an HTTP request and receives a response with the version of the required JTAPI API.

The application compares the version that is available on the server to the local version in the application classpath and determines whether an upgrade is necessary. This allows applications to refresh the jtapi.jar component to match the Cisco Unified Communications Manager and provides a way to centrally deploy the jtapi.jar to which applications can auto update.

The API that is required to perform this functionality gets packaged in the form of an updater.jar. The jtapi.jar and updater.jar get packaged with the standard manifest, which can be used to compare versions.

Note This feature does not update JTAPI Preferences, JTAPITestTools, Updater.jar and javadoc components. If applications require these components, install JTAPI from the Cisco Unified Communications Manager plug-in pages. Auto Update supports JTAPI Release 2.0 and later.

Refer to Cisco Unified JTAPI Installation, on page 213 for more information.

The following new or changed interfaces exist for autoupdate of APIs:

Class com.cisco.services.updater.ComponentUpdater

Component	`queryLocalComponentVersion (java.lang.String componentName, java.lang.String path)` Throws an IOException, IllegalArgumentException.
Component	`queryServerComponentVersion (java.lang.String componentName, java.lang.String urlString)` Throws an IOException, IllegalArgumentException, and sends an HTTP query to the server to determine the remote server installed components version.

Interface com.cisco.services.updater.Component

int	`compareTo` **`(Component otherComponent)`**
Component	`fetchFromServer` **`()`** Performs an HTTP fetch of the component from the server and writes to the local file system with the file name temp.jar in the local directory.
java.lang.String	`getBuildDescription ()` Returns the string 'Release' for a version of the form 'a.b(c.d) Release'.
int	`getBuildNumber` **`()`** Returns 'd' for a version of the form a.b(c.d).
java.lang.String	`getLocation ()` The string form location of the component.
int	`getMajorVersion` **`()`** Returns 'a' version for a version of the form a.b(c.d).
int	`getMinorVersion` **`()`** Returns 'b' version for a version of the form a.b(c.d).
java.lang.String	`getName ()` Returns the name of the component.
int	`getRevisionNumber` **`()`** Returns 'c' for a version of the form a.b(c.d).

The Autoupdater feature in JTAPI also allows applications to download the latest version of JTAPI.JAR directly from the Cisco Unified Communications Manager.

1. Updater creates a newjtapi.jar file in the current folder of the application, which represents the new version of the jar file that was downloaded from the Cisco Unified Communications Manager.
2. Updater copies the current jtapi.jar to a file that is named component.temp in the classpath specified.
3. Updater replaces the current jtapi.jar file with the new jtapi.jar file.

At the end of this operation, the current jar file becomes component.temp and the new jar file becomes jtapi.jar. This operation is supported for both Linux and Windows.

Example Usage of Autoupdater

```
Command Line : java com.cisco.services.updater.ComponentUpdater <server> <component name>
<login> <passwd>Component localComponent,  downloadedComponent;
ComponentUpdater updater  =  new ComponentUpdater();
String localPath  =  updater.getLocalComponentPath(args[1]);
localComponent  =  updater.queryLocalComponentVersion("jtapi.jar",  localPath);
localComponent.copyTo("component.temp");
String provString  =  args[0] + ";login = " + args[2] + ";passwd = " + args[3];

CiscoJtapiPeer peer  =  (CiscoJtapiPeer) (JtapiPeerFactory.getJtapiPeer(null));
CiscoJtapiProperties tempProp  =  ( (CiscoJtapiPeerImpl) (peer)). getJtapiProperties();
tempProp.setLightWeightProvider(true);
```

```
Provider provider  =  peer.getProvider(provString);
String url  =  ( (CiscoProvider) (provider)).getJTAPIURL(); provider.shutdown();
Component serverComponent  =  updater.queryServerComponentVersion("jtapi.jar",  url);

downloadedComponent  =  serverComponent.fetchFromServer();
int retVal  =  downloadedComponent.replaces(localComponent);
```

The “replaces” API will replace the existing JTAPI version with the new version.

Note The updater will only update the JTAPI.JAR file and not the other sample applications and Cisco JTAPI documentation that are bundled with the JTAPI plug-in. To get these other components, applications must download the plug-in from the Cisco Unified Communications Manager and install it.

Barge and Privacy Event Notification

The Barge Feature provides the ability for shared addresses to barge into an established call of address on another terminal. This feature gets activated when an address TerminalConnection is in the passive state and CallCtlTerminalConnection is in the bridged state. This version of Cisco Unified JTAPI only supports feature activation manually on application-controlled terminals (IP phones). For this release, you cannot activate the feature through an API.

The Privacy feature provides the ability to enable or disable other shared addresses to barge into call. When privacy is enabled, other shared addresses cannot barge into a call and vice versa. Privacy represents a terminals property. IP phones have a “Privacy” softkey and pressing it enables or disables the privacy. Privacy can be dynamically enabled or disabled for the active calls on the terminal. When privacy is on for the call, the TerminalConnection for the call appearances on the shared address appear in the “InUse” state. If privacy status changes during the CallProgress, CiscoTermConnPrivacyChangedEvent gets delivered to the application.

Two types of barge feature functionalities exist in Cisco Unified Communications Manager: one uses built-in conference bridge called “Barge, ” while another uses shared conference bridge resources called “CBarge”. From the application point of view, no interface changes exists between Barge and CBarge; however, some behavioral changes, which are described in the message flow diagram in Message Sequence Charts, on page 755 occur.

Barge, CBarge, and Privacy have these interfaces:

Interface CiscoTerminalConnection.getPrivacyStatus()

`booleangetPrivacyStatus()`

This interface returns the privacy status of a call on the terminal.

Interface CiscoTermConnPrivacyChangedEv

`javax.telephony.TerminalConnectiongetTerminalConnection()`

A new reason code, CiscoCall.CAUSE_BARGE gets added to CiscoCall for barge events.

JTAPI provides CallCtiCause as CiscoCall.CAUSE_BARGE when a SharedLine TerminalConnection or CallCtiTerminalConnection goes to an active or talking state as a result of barge. This cause code also gets provided in CallCtiEvents for dropping temporary calls that are created during the barge operation.

This cause code is not provided for the CBarge scenario.

For details on these interfaces, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for barge, CBarge, and privacy, see Message Sequence Charts, on page 755.

Call Control Discovery

The Call Control Discovery (CCD) feature facilitates provisioning for inter-call agent communications. It uses the Service Advertisement Framework (SAF) network service to advertise itself as a call control entity and to discover other call control entities (Cisco Unified Communications Managers or CMEs) on the network so that it can dynamically adapt their routing behavior.

When a call is made between two devices on different clusters and the call is rejected with a cause code other than unallocated , unassigned number and user busy, the CCD feature fails over the call to a PSTN network. That is, the call is routed through a PSTN network instead of an IP network to reach the same destination.

JTAPI supports the SAF CCD feature. However, applications are not notified when a normal SAF call fails over to a PSTN trunk.

JTAPI exposes a new reason CiscoFeatureReason.REASON_SAF_CCD_PSTN_FAILOVER for the new connection created for the redirect or forward destination. This occurs when there is a redirect or forward across the cluster through an SAF trunk and the call fails over to a PSTN trunk.

Interface Changes

See CiscoFeatureReason, on page 402

Message Sequences

See Call Control Discovery, on page 779

Backward Compatibility

This feature is backward compatible.

Call Forward

Cisco Unified JTAPI supports setting the Call Forward feature according to the JTAPI Specification. Cisco Unified JTAPI implementation does not support all the forwarding characteristics but supports only the FORWARD_ALL attribute for the Address. Applications can invoke setForwarding, getForwarding, and cancelForwarding methods on a CallControlAddress object, but the CallControlForwarding instruction can only be of type FORWARD_ALL.

Call Forward Override

This feature provides a mechanism to override the call forward all feature. If a user (CFA Initiator) sets CFA to another user (CFA target), the CFA should be ignored if the CFA target calls the CFA initiator. This would allow the CFA Target to reach the CFA Initiator for important calls.

The behavior of this CallManager feature is configurable via service parameter - CFADestinationOverride.

Example: Alice has a phone with DN 1000 * Bob has a phone with DN 2000 * Daniel has a phone with DN 4000 * Alice does a CFA to 2000

CFA behavior * Bob calls Alice. Call goes to Alice and does not follow CFA back to himself. * Daniel calls Alice. Call follows CFA to Bob. * Bob answers and transfers the call to Alice. Bob can do this because Alice has her phone forwarded to Bob. There is no interface change to JTAPI layer with this feature. However JTAPI applications could perceive a difference in behavior when CiscoAddress.setForward() API is invoked. In scenario where CFA target calls the CFA initiator as described in example, call is not forwarded if feature is enabled.

Backward Compatibility

JTAPI applications that were written for Release 5.0 should be backward compatible with Release 5.1. JTAPI Client Upgrade Application does not require JTAPI Client upgrade to run or be backward compatible. JTAPI Client upgrade is required only if new features are used.

Call Park

Cisco Unified JTAPI supports user interactions with Call Park and reports the appropriate events to the applications. When a call is parked from an IP phone, the connection that belongs to the parking address moves into Disconnected state, and the associated TerminalConnection moves into Dropped state. A new connection in queued state for the park number gets created.

If an application is monitoring only the address that parked the call, all existing connections get Disconnected, TerminalConnections get Dropped, and the call moves to Invalid state.

Call Pickup

Call Pickup enables devices to receive alerts within Call Pickup Groups and events, to act on these alerts by invoking APIs that support variants of Call Pickup.

These APIs allow applications to gather information about existing Call Pickup groups, and register and unregister for receiving pickup alerts for specific pickup groups.

JTAPI supports invoking Pickup, Group Pickup, Other Pickup, and Directed Call Pickup from applications. In Cisco Unified Communications Manager releases prior to release 8.0(1), all these features except Other Pickup were supported as observed events, but were not invoked.

Note Call Pickup is not supported on CTI route points.

Interface Changes

CiscoPickupGroup, on page 472, CiscoAddress, on page 283, CiscoTerminal, on page 611, CiscoProvider, on page 486, CiscoProviderCapabilities, on page 498, CiscoProvPickupCallAlertEv, on page 481, ProviderPickupNotificationRegistrationClosedEv, on page 665, CiscoAddrPickupGroupChangedEv, on page 307.

Message Sequences

Call Pickup, on page 1103

Backward Compatibility

This feature is backward compatible.

Call Recording for SIP or TLS Authenticated Calls

Prior to 12.5(1) version, the phones which are authenticated (phone with Security profile having Device Security Mode as Authenticated) were not allowed to make use of the Call Recording feature. Whereas, Non–Secured phones or Secured/ Encrypted phones could use Call Recording feature with Non-Secured or Secured recorders, respectively. With the release 12.5(1), Cisco Unified CM JTAPI interface has been enhanced to allow recording in Authenticated Phones based on the value of the new service parameter **Authenticated Phone Recording**.

The expectation is that the authenticated phones should also be allowed to make use of the Call Recording feature. It depends on value set in the newly added service parameter **Authenticated Phone Recording** which can be set to the following values:

- **Allow Recording** – Authenticated Phones can be allowed to record the calls.
- **Do Not Allow Recording** – Authenticated Phones cannot make use of Call Recording feature. This is the default value for the service parameter. The behavior would be the same as that of the current behavior.

Backward Compatibility

This feature is backward compatible. JTAPI will support the current API's.

Call Select Status

Cisco Unified JTAPI sends CiscoTermConnSelectChangedEv event whenever the call is selected either by feature or by manually. Once application receives the event, application can use TerminalConnection.getSelectStatus() to get proper call select status. There are three possible statuses by calling TerminalConnection.getSelectStatus() as follows:

- CiscoTerminalConnection. CISCO_SELECTEDNONE: The select status means that the call is not selected
- CiscoTerminalConnection. CISCO_SELECTEDLOCAL: The select status means that the call is selected on the terminal connection
- CiscoTerminalConnection. CISCO_SELECTEDREMOTE: Passive TerminalConnection will get this select status if the call is selected by it's shared line

Backward compatibility

This feature is not backward compatible.

Calling Party Display Name

The CiscoCall interface provides methods to get name displays of the calling party and the called party in a call. Applications can use getCurrentCallingPartyDisplayName() to get the display name of the calling party.

JTAPI applications can use the following interface to get the display names of the calling party and the called party.

```
{..
..
/**
 *This interface returns the display name of the called party in the call.
 *It returns null if display name is unknown.
*/
public String getCurrentCalledPartyDisplayName();

/**
 *This interface returns the display name of the calling party.
 *It returns null if display name is unknown.
*/
public String getCurrentCallingPartyDisplayName();
}
```

The address objects store the display name internally, and the name gets updated when currentCallingAddress and currentCalledAddress are updated. NULL returns if the call is not in the active state and if currentCalling and currentCalled addresses of the call are not initialized.

Note The system does not support Call.getCurrentCalledAddress() and call.getCurrentCallingAddress() for conference calls. Also, the system does not support call.getCurrentCalledPartyDisplayName() and call.getCurrentCallingPartyDisplayName() for a conference call.

Calling Party IP Address

Extensions to CallCtlConnOfferedEv and RouteEvent provide a method for retrieving the IP address of the calling party. This feature provides the calling party IP address to the destination side of basic calls, consultation calls for transfer and conference, and basic redirect and forwarding. The system does not support other scenarios and feature interactions, including those where the calling party changes. This feature only supports IP phones as calling party devices, although IP address of other calling devices may also be provided. See CiscoCallCtlConnOfferedEv, on page 344 and CiscoRouteEvent, on page 516.

Backward compatibility

This feature is backward compatible.

Calling Party IP Address

The Calling Party IP Address enhancement provides the calling party IP address to the destination side of basic calls, consultation calls for transfer and conference, and basic redirect and forwarding. Only calling party IP phones are supported, although IP address of other calling devices may also be provided.

Note Other feature interactions are not supported including those during which the calling party changes.

New Cisco extensions to the CallCtlConnOfferedEv and RouteEvent classes are created and expose a method to obtain the calling party IP address. The new extensions are CiscoCallCtlConnOfferedEv and CiscoRouteEvent. An empty returned value indicates that the calling party IP address is not available.

Basic Call scenario

JTAPI application monitors party B

Party A is an IP phone

A calls B

IP Address of A available to JTAPI application monitoring B consultation transfer scenario

JTAPI application monitors party C

Party B is an IP phone

A talks to B

B initiates a consultation transfer call to C

IP Address of B is available to JTAPI application monitoring party C

Consultation conference scenario

JTAPI application monitors party C

Party B is an IP phone

A talks to B

B initiates a consultation conference call to C

IP Address of B is available to JTAPI application monitoring party C

Redirect scenario

JTAPI application monitors party B and party C

Party A is an IP phone

A calls B

IP Address of A is available to JTAPI application monitoring party B

Party A redirects B to party C

Calling IP address is not available to JTAPI application monitoring party B

Calling IP address of B is provided to JTAPI application monitoring party C

Backward compatibility

This feature is backward compatible. Application must invoke a new API to query IP address of a call.

Calling Party Normalization

Calling Party Normalization (CPN) is an enhancement. This feature provides the option to transform or normalize the incoming call number and convert into the E.164 format, which includes the (country code, state code, and number type). The number type field identifies the subscriber, national, international, or unknown. The number type is not supported in conference scenarios.

Interface changes

This feature introduces a new method in CiscoCall that is getGlobalizedCallingParty() and a new method in CiscoPartyInfo that is getNumberType(). See CiscoCall, on page 326 and CiscoPartyInfo, on page 470 for more information.

Message sequences

See Calling Party Normalization, on page 1079

Backward compatibility

This feature is backward compatible.

CallFwdAll Key Press Notification

This feature enables applications to know whether the call is a normal call or a temporary call, when the CallFwdAll key is enabled.

JTAPI exposes this information through the API getCFwdAllKeyPressIndicator() which is exposed on the CiscoCall interface. This API enables the application to know if the call is created due to pressing of CallFwdAll softkey or not. The newly added getCFwdAllKeyPressIndicator()" could return following constants that are also new:

- If it is pressed on a phone that is in on-hook state to set CallFwdAll, this API returns CiscoCall.CFWD_ALL_SET.
- If it is pressed on a phone that is in on-hook state to clear the CallFwdAll, this API returns CiscoCall.CFWD_ALL_CLEAR.
- If the call is made first and then the user presses CallFwdAll key when phone is in off-hook state, this API returns CiscoCall.CFWD_ALL_NONE.

  Interface changes

See CiscoCall, on page 326

Message Sequences

See CallFwdAll Keys Press Notification, on page 787

Backward Compatibility

This feature is backward compatible.

CallSelect and UnSelect Event Notification

You can select or unselect call on a phone for doing DirectTransfer or join or any other feature operation. When a SharedLine user selects a call, the RemoteInUse shares line TerminalConection will go passive, and CallCtlTermiCallConnection goes in InUse state. When call is unselected, CallCtlTerminalConnection goes into a bridged state. An application cannot invoke any API on Passive/InUse TerminalConnection. CallProcessing also performs a Select/UnSelect operation during features (such as transfer/conference) operation. Applications will also perceive these events if the applications monitor RemoteInUse terminal.

For example, if A and A' are SharedLine, and A selects the call, CallCtlTerminalConnection of A' goes into a passive or InUse state. If A “UnSelects” the call, the CallCtlTerminalConneciton of A' goes into the passive or bridged state.

To view the message flow for CallSelect or UnSelect, see Message Sequence Charts, on page 755

Certificate Download API Enhancement

Currently Cisco Unified JTAPI certificate download API has some security issues, to solve the problem, Cisco provides new certificate download APIs. New APIs require applications to specify a certificate pass phrase and the certificate pass phrase is used to encrypt Java key store where client/server certificates are stored.

Old certificate download APIs are deprecated, however, it will still remain for some time to avoid backward compatibility issue for applications. Cisco highly recommends to migrate the application to new APIs.

Cisco Unified JTAPI also provides new API deleteCertificate() and deleteSecurityPropertyForInstance() that can be used by application to delete certificates already installed. To change pass phrase for certificate java key store, the application must delete the old certificate by using this API and upload new certificate.

JTAPIPreferences UI security tab enhancement provides two new buttons, one for DeleteCertificate and another for Update Certificate. DeleteCertificate button allows users to delete the certificate for required username/instanceID. Update Certificate button allows users to upload the certificate from CAPF server. If certificate update is successful, certificate update box is updated to show Updated; authorization string and certificate pass phrase are cleared. If certificate update operation fails, certificate box continues to show status Not Updated status unless certificate was previously updated. User/Applications must provide certificate pass phrase every time they try to update certificate, Cisco Unified JTAPI does not save certificate pass phrase for security reason in any circumstances. Applications own the responsibility to secure the pass phrase and provide it through API when needed.

Backward Compatibility

This feature is backward compatible.

Changes in DeviceType Name Handling

Currently, TSP hardcodes the DeviceTypeName depending on the DeviceType. When a new device type is added, we have to manually add the new device type name to the list of supported devices. Because CTI does not fetch and store the device type name in its cache, TSP cannot get this info from CTI. TSP needs to update the device type name when a new device type is added without any manual intervention.

In JTAPI, the changes have been made to ensure that QBE interface changes to handle the receive devicetypename that is sent from CTI and is stored in the deviceInfo structure. It is not used anywhere in JTAPI and will not be exposed to applications. Only the QBE interface changed as follows:

```
public DeviceRegisteredEvent ( String ride,  int deviceType,  boolean
allowsRegistration,  int deviceID,  boolean loginAllowed,  UnicodeString userID,
  boolean controlled,  int reasonInt,  int registrationType,  int unicodeEnabled,
  int locale,
// added for deviceTypeName change
String devTypeName) {
public DeviceUnregisteredEvent ( String deviceName,  int deviceType,  boolean
allowsRegistration,  int deviceID,  UnicodeString userID,  boolean
controllableBool,  int reasonInt ,  int locale,
//added for devtypename support
String devTypeName) {
```

Cisco MediaTerminal

In JTAPI, the terminal object represents the logical endpoint for a call and is presumed to be able to receive and transmit data (digital encoded voice samples, for example). Thus, terminals in JTAPI represent Cisco Unified IPPhones. Even though gateways terminate media, terminals do not represent them. The CiscoMediaTerminals in particular represent a special kind of endpoint for which applications take responsibility for media termination.

The following four steps associate with using CiscoMediaTerminals:

- Provisioning
- Registration
- Adding Observers
- Accepting Calls

Provisioning

Ensure CiscoMediaTerminals, which are analogous to physical terminals, get provisioned accordingly in Cisco Unified Communications Manager, even though they do not represent actual hardware IP phones or gateways. Just as IP phones must be added to Cisco Unified Communications Manager database by using the Device Wizard, CiscoMediaTerminals get added the same way, so Cisco Unified Communications Manager can associate the application endpoint with a directory number and other call control properties such as call forwarding. No device type called CiscoMediaTerminal exists in the DeviceWizard. Instead, Cisco Unified Communications Manager has one or more device types that support application registration—each of these types get exposed as a CiscoMediaTerminal through JTAPI. Currently, only the device type CTI port represents a CiscoMediaTerminal in JTAPI.

This procedure lists the steps for provisioning a CTI port for use as an application-controlled endpoint.

1. Within the Cisco Unified Communications Manager configuration windows, add a CTI port device from the Device-Phone window by using the Device Wizard. The CTI port device name specifies the name of the corresponding CiscoMediaTerminal in JTAPI.
2. Add the new CTI port device, by using the User-Global Directory window, to the list of devices that the application controls by using the User window.

For more information, refer to the Cisco Unified Communications Manager Administration Guide.

Registration

After a media termination device is properly provisioned in Cisco Unified Communications Manager, the application may obtain a reference to the corresponding CiscoMediaTerminal object by using either the Provider.getTerminal() method or CiscoProvider.getMediaTerminal() method. The two methods differ in that the CiscoProvider.getMediaTerminal() method only returns CiscoMediaTerminals, whereas Provider.getTerminal() will return any terminal object that is associated with the provider, including those representing physical IP phones.

Use the CiscoMediaTerminal.register() method to notify Cisco Unified Communications Manager of the intent to terminate RTP streams of certain payload types. The CiscoMediaTermina.register() method takes an IP address, a port number, and an array of CiscoMediaCapability objects that indicate the types of codecs that the application supports as well as codec-specific parameters.

The IP address and port indicate the address where the application can receive media streams. The following sample code demonstrates how to register a CiscoMediaTerminal and bind it to a local address, port number 1234:

```
CiscoMediaTerminal registerTerminal (Provider provider,  String terminalName) {
 final int PORT_NUMBER  =  1234;
  try {
   CiscoMediaTerminal terminal  =  provider.getTerminal (terminalName);
   CiscoMediaCapability [] caps  =  new CiscoMediaCapability [1];
   caps[0]  =  CiscoMediaCapability.G711_64K_30_MILLISECONDS;
   terminal.register (InetAddress.getLocalHost (),  PORT_NUMBER,  caps);
  }
  catch (Exception e) {
    return null;
  }
}
```

For this sample code to work, ensure the specified provider is IN_SERVICE. Further, be aware that this code uses the constant CiscoMediaCapability.G711_64K_30_MILLISECONDS. This actually represents a static reference to a CiscoG711MediaCapability object that specifies a 30-millisecond maximum RTP packet size. The CiscoMediaCapability class predefines this and other common media formats.

To specify a media payload that is not listed in the CiscoMediaCapability class, two options exist. If the desired payload type is a simple variation of one of the existing subclasses of CiscoMediaCapability, you only need to construct a new instance of the subclass. For instance, if an application can support G.711 payloads with a 60-millisecond maximum RTP packet size, it can construct the CiscoG711MediaCapability object directly; including specifying 60 milliseconds in the constructor.

Alternatively, if no existing subclass of CiscoMediaCapability that matches the desired payload type exists, construct an instance of the CiscoMediaCapability class directly. The maximum packet size, for example, 30-milliseconds, represents the only other parameter that may be specified when a CiscoMediaCapability is constructed.

The following code illustrates registering a custom payload capability:

```
CiscoMediaTerminal registerTerminal (Provider provider,  String terminalName) {
 final int PORT_NUMBER  =  1234;
  try {
   CiscoMediaTerminal terminal  =  provider.getTerminal (terminalName);
   CiscoMediaCapability [] caps  =  new CiscoMediaCapability [1];
```

```
   caps[0]  =  new CiscoMediaCapability (
     RTPPayload.G728,
     30  // maximum packet size,  in milliseconds
     );
   terminal.register (InetAddress.getLocalHost (),  PORT_NUMBER,  caps);
  }
  catch ( Exception e) {
    return null;
  }
}
```

The payload type parameter that is used for constructing the CiscoMediaCapability object corresponds to the payload field in the RTP header. The RTPPayload interface defines a number of well-known payload types for this purpose.

Adding Observers

To receive events that indicate where and when to transmit and receive RTP data, place a CiscoTerminalObserver on the CiscoMediaTerminal. The CiscoTerminalObserver extends the standard JTAPI TerminalObserver interface without defining any new methods; it provides a marker interface that signals the application interest in receiving RTP events.

Note Because this is a TerminalObserver, not a CallObserver, it must get added by using the Terminal.addObserver() method, not the Terminal.addCallObserver() method.

Additionally, add a CallControlCallObserver to the Address object that is associated with the CiscoMediaTerminal. This guarantees that the application will get notified when calls are offered to the CiscoMediaTerminal. Unlike regular IP phones, which automatically accept any offered call, CiscoMediaTerminals accept, disconnect (reject), or redirect any call that is offered to it. Because the CallCtlConnOfferedEv only gets presented to CallControlCallObservers that are placed on Address objects, not Terminal objects, the application places its CallControlCallObserver in the correct place.

Note Be sure to implement the CallControlCallObserver interface, not just the CallObserver interface; the CallCtlConnOfferedEv will not get delivered to observers that implement only the core CallObserver interface.

Accepting Calls

When an inbound call arrives at the CiscoMediaTerminal address, it must be accepted by using the CallControlConnection.accept() method before a terminal connection gets created. This process does not apply for outbound calls —the connection will occur in the CallControlConnection.ESTABLISHED state as soon as the call progresses beyond digit recognition. After the connection is accepted, answer the ringing terminal connection to start media flow. Assuming that Cisco Unified Communications Manager can match the capabilities that were registered with the capabilities of the calling endpoint, Cisco Unified Communications Manager sends the Media Flow events, so the application can begin transmitting and receiving RTP data.

Cisco Unified Communications Manager Media Endpoint Model

Endpoints represent the entities within the Cisco Unified Communications Solutions platform that terminate media, such as IP telephones and gateways. A call from one endpoint to another results in media flowing between the two endpoints. All endpoints in the Cisco Unified Communications Solutions platform transmit voice data by using real-time protocol (RTP). The Cisco Unified Communications Solutions telephones and gateways, for example, include built-in RTP stacks. Applications may also act as endpoints in a Cisco Unified Communications Solutions system; that is, they may terminate media. Because all Cisco Unified Communications Solutions endpoints use RTP, applications also must be able to transmit and receive RTP packets.

Payload and Parameter Negotiation

In addition to bearer data and payload, each RTP packet contains a header that helps endpoints to determine how to reassemble and decode a sequence of such packets into a media stream. RTP does not provide, however, a means for endpoints to negotiate which payload type to use for a particular stream: for example, audio data that is encoded by using the G.711 standard. Furthermore, RTP does not offer a means of negotiating unique payload type parameters such as the sampling rate of the encoded data or the number of samples that are to be transferred in each RTP packet. Instead, RTP usually gets used in conjunction with another protocol such as H.323, which specifies its own method for endpoints to negotiate these parameters. All such negotiation occurs prior to transmitting RTP packets between endpoints.

Cisco Unified Communications Manager, not the endpoints, has responsibility for selecting the payload and encoding parameters for RTP streams. The following five steps that are involved in a typical bidirectional audio telephone call apply:

- Initialization
- Payload Selection
- Receive Channel Allocation
- Starting Transmission and Reception
- Stopping Transmission and Reception

Initialization

Upon startup, each endpoint informs Cisco Unified Communications Manager of its media capabilities, that is, G.711, G.723, G.729a, and so on. Startup for an IP phone, for example, occurs when the phone is first turned on, or after it recontacts Cisco Unified Communications Manager after losing its former connection. The endpoint cannot express a preference for one payload type versus another, but it can specify certain parameters for each payload type, such as, packet size.

The capability list that the endpoint registers remains exclusive and immutable. If the endpoint specifies that it can support both G.711 and G.723, it implicitly declares that it cannot support G.729a. Moreover, the endpoint must disconnect from Cisco Unified Communications Manager and reinitialize to change the list of capabilities that it supports.

JTAPI applications perform this step by registering a CiscoMediaTerminal with Cisco Unified Communications Manager. The CiscoMediaTerminal.register() method allows applications to supply an array of media capability

objects for registration with Cisco Unified Communications Manager. This step informs Cisco Unified Communications Manager that the application will act as the endpoint for all calls to or from a particular directory number, as determined by the device configuration in the Cisco Unified Communications Manager configuration.

Payload Selection

When a bidirectional media stream is about to be created between two endpoints, for instance, when a call is answered at an endpoint, Cisco Unified Communications Manager selects an appropriate payload type (codec) for the media stream. Cisco Unified Communications Manager compares the media capabilities of both endpoints that are involved in the call and selects the appropriate common payload type and payload parameters to use.

The basis for payload selection includes endpoint capabilities and location, although other criteria may get added to this selection logic in the future. Endpoints do not get dynamically involved in selecting payload types on a call-by-call basis.

Receive Channel Allocation

If Cisco Unified Communications Manager can find a common payload type for the RTP stream between the two endpoints, it requests that each endpoint create a logical “receive channel”; that is, a unique IP address and port at which the endpoint will receive RTP data for the call. Each endpoint returns an IP address and port to Cisco Unified Communications Manager in response to this request.

Currently, only IP phones and gateways perform this step. Cisco Unified Communications Manager requires JTAPI applications to specify a fixed IP address and port during initialization. Therefore, JTAPI applications cannot terminate more than one media stream simultaneously for the same endpoint. Applications that want to terminate multiple media streams must register multiple endpoints simultaneously.

If the endpoint does not respond to the open receive channel request quickly enough, Cisco Unified Communications Manager disconnects the call. Because JTAPI applications always supply an IP address when CiscoMediaTerminals are registered, calls to application-controlled endpoints do not get disconnected for this reason. However, if Cisco Unified Communications Manager cannot find a common payload type between the two endpoints that are involved in the call, Cisco Unified Communications Manager disconnects the call.

Starting Transmission and Reception

After Cisco Unified Communications Manager receives channel information for both parties, it informs each endpoint of the codec parameters that it selected for the RTP stream and the destination address for the other endpoint. This information gets conveyed in two messages to each endpoint: a start transmission message and a start reception message.

JTAPI applications receive the CiscoRTPOutputStartedEv and CiscoRTPInputStartedEv events that contain all the codec parameters that are necessary for sending and receiving RTP data.

As a part of the QoS baselining effort in JTAPI, CiscoRTPOutputStartedEv provides the getPrecedenceValue() API to applications. CTI presents this value, The DSCP for Audio Calls to JTAPI. Using this value, applications can set the DSCP value for the media streams that they open.

Stopping Transmission and Reception

When the RTP stream must get interrupted because of a feature such as hold or disconnect, Cisco Unified Communications Manager requests that each endpoint stop its transmission and reception of RTP data. Just as when the media flow is started, the stop transmission and stop reception messages get sent separately.

JTAPI applications receive the CiscoRTPOutputStoppedEv and CiscoRTPInputStoppedEv.

Cisco Unified Communications Manager Server Failure

If a Cisco Unified Communications Manager server fails, the associated devices re-home to the next Cisco Unified Communications Manager server in the group. The prioritized list of Cisco Unified Communications Managers in the device pool information configuration for each device defines this process.

Failure of a Cisco Unified Communications Manager server only results in a partial outage of devices in the cluster. Those devices remain available following a successful Cisco Unified Communications Manager failover and registration with a secondary Cisco Unified Communications Manager.

Note A device such as a Cisco Unified IPPhone 7960 fails over to a secondary Cisco Unified Communications Manager server only when no active calls exist on that device. The failure of a Cisco Unified Communications Manager server during a call results only in termination of observation of that device. The media path continues to exist but without any further call control features.

Cisco Unified JTAPI communicates this partial outage to applications by using CiscoAddrOutOfServiceEv and CiscoTermOutOfServiceEv events. When the Cisco Unified Communications Manager fails over, the device must successfully register to the secondary Cisco Unified Communications Manager before the device is available to the JTAPI applications. Cisco Unified JTAPI will send the CiscoAddrInServiceEv and CiscoTermInServiceEv events.

The Provider remains in service during this time. Devices on other Cisco Unified Communications Manager servers remain available for call control. The events get sent on callbacks of the respective Address or Terminal observer objects. CiscoAddrOutOfServiceEv and CiscoAddrInServiceEv events get sent to an object that is implementing the AddressObserver and get added to an Address by using the addressChangedEvent() callback object method. The CiscoTermOutOfServiceEv and CiscoTermInServiceEv events get sent to an object that is implementing the TerminalObserver interface and get added to a Terminal that is using the terminalChangedEvent() callback method.

If the devices are currently in a call, a CallObservationEnded message is sent on the CallObserver callChangedEvent() callback, followed by the CiscoAddrOutOfServiceEv and CiscoTermOutOfServiceEv messages.

Note Applications must monitor for and respond to the CiscoAddrOutOfServiceEv, CiscoTermOutOfServiceEv, CiscoAddrInServiceEv, and CiscoTermInServiceEv events before the calling call control functions on the address or terminal. Applications that do not support this action may encounter unexpected errors because the applications do not know the exact state of the system.

Cisco Unified IP 7931G Phone Interaction

You can configure Cisco Unified IP 7931G phones in two modes:

- NoRollOver
- RollOver (across the same DN or across different DNs)

When Cisco Unified IP 7931G phones are configured in NoRollOver mode, they operate like regular phones that are running SCCP, and in this mode transfers or conferences cannot occur across the different addresses. JTAPI will support controlling and monitoring of a 7931G phone when it is configured in NoRollOver mode.

In RollOver mode, Cisco Unified IP 7931G phones support transfer or conference across different addresses. In this mode, JTAPI does not allow controlling and monitoring of a Cisco Unified IP 7931G phone. Applications see such terminal/addresses as restricted. If a Cisco Unified IP 7931G phone is in the control list of an application user and the phone configuration changes from NoRollOver to RollOver mode, JTAPI sends a CiscoAddrRestrictedEv event for addresses on the Cisco Unified IP 7931G phone and sends a CiscoTermRestrictedEv for terminals, with cause CiscoRestrictedEv.CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION.

However, if the phone configuration changes from RollOver to NoRollOver mode, JTAPI sends a CiscoAddrActivatedEv event for addresses on the Cisco Unified IP 7931G phone and sends a CiscoTermActivatedEv for terminals.

If a Cisco Unified IP 7931G phone that is configured in RollOver mode transfers or conferences to JTAPI-controlled addresses, JTAPI applications do not recognize a common controller in the final and the consult call. This would provide different behavior to the JTAPI application. Depending on how the JTAPI application is processing information that is provided in events, applications may require changes to handle JTAPI events for this transfer or conference scenario.

You can disable transfers and conferences across the line by configuring the Cisco Unified IP 7931G phone to NoRollOver mode through the phone configuration window of Cisco Unified Communications Manager Administration.

There are two new cause codes for the CiscoRestrictedEv interface. When the terminal or address is restricted because a Cisco Unified IP 7931G phone is configured in RollOverMode, JTAPI sends CiscoAddrRestrictedEv with cause CiscoRestrictedEv.UNSUPPORTED_DEVICE_CONFIGURATION. This release also introduces a default cause code CAUSE_UNKNOWN, which applications should handle.

Backward Compatibility

This feature is backward compatible. You can disable this feature by configuring all Cisco Unified IP 7931G phones in a cluster in NoRollOver mode or by not having any Cisco Unified IP 7931G phones in a Cisco Unified Communications Manager cluster. If any phone in a Cisco Unified Communications Manager cluster is configured with RollOver mode, it may cause changes to the behavior of JTAPI-controlled addresses and terminals.

For more information, see CiscoRestrictedEv, on page 513.

Cisco Unified JTAPI Install Internationalization

Cisco Unified JTAPI supports multiple languages for the JTAPI installation and user preference UI. When JTAPI launches, you receive options for choosing languages for the installation. After choosing a language, further installation instructions display in the chosen language. The first option always specifies English. If certain phrases are missing in the locale language, the instructions default to English. See Cisco Unified JTAPI Installation, on page 213 for more information.

Cisco VG248 and ATA 186 Analog Phone Gateways

Cisco Unified JTAPI supports control of analog phones that are connected to the Cisco VG248 and ATA 186 Analog Phone Gateways. By adding the Cisco VG248 and ATA 186 Analog Phone Gateways to the user-controlled list, applications can control the devices.

Applications receive events for the devices in a way similar to other IP phones. Applications can also initiate calls and invoke other features except answer Request through APIs. Make call works only when the device goes physically off hook.

Applications cannot answer calls from APIs for the devices. If an application attempts to answer () on TerminalConnection for the VG248 and ATA 186 Terminal, the system throws PlatformException with error CiscoJtapiException.COMMAND_NOT_IMPLEMENTED_ON_DEVICE. To answer calls, you must manually pick up the handset, and then you can invoke other call control features such as transfer, conference, blind transfer, and park from the API.

CiscoJtapiExceptions

Cisco Unified JTAPI notifies the application of CTI-generated error codes. These codes return when an exception or error occurs in the CTIManager. The CTI returned error code propagates to the application separately. The application can extract the error code by invoking getErrorCode() method on the exception object, can get CTI error code name by invoking getErrorName() method, and can get error description by invoking method getErrorDescription().

The methods getErrorName(int errorCode) and getErrorDescription (int errorCode) deprecate and get removed in future releases. Cisco recommends that application users do not use these methods.

CiscoJtapiExceptions interface defines error codes in JTAPI.

When a PlatformException is thrown, it can be queried to get the error code, which can be compared to the following.

Errors

Problem Error Message `CTIERR_LOGIN_FAILED_PWD_EXPIRED_USER_CAN_RESET`

Possible Cause This value is a static definition that identifies an error code as a login failure due to an expired password. In addition, this error code lets the application know that the user can change their password.

Solution Try resetting the password.

Problem Error Message `CTIERR_LOGIN_FAILED_PWD_EXPIRED_USER_CANNOT_RESET`

Possible Cause Explanation This value is a static definition that identifies an error code as a login failure due to an expired password. In addition, this error code lets the application know that the user cannot change their password, and that an administrator will have to reactivate the account.

Solution Contact the administrator to reactivate the account.

Problem Error Message `CTIERR_LOGIN_FAILED_ACCOUNT_LOCKED`

Possible Cause This value is a static definition that identifies an error code as a login failure due to the user account being locked. This is a generic exception for the various types of account lockout. The applications are not informed the reason for the account lockout.

Solution Contact the administrator to unlock the account.

Problem Error Message `CTIERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING`

Possible Cause This error code is returned when an application invokes a stopRecording() request and passes a method of recording other than the method that was specified when the recording was started.

Problem Error Message `CTIERR_INVALID_REMOTE_DESTINATION_NUMBER`

Possible Cause This error code is returned when an invalid remote destination number is enterred.

Problem Error Message `CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER`

Possible Cause This error code is returned when the same remote destination number is entered twice.

Problem Error Message `CTIERR_REMOTEDESTINATION_LIMIT_EXCEEDED`

Possible Cause This error code is returned when the number of remote destinations has exceeded the max number limit.

Problem Error Message `CTIERR_REMOTE_DEVICE_REQUEST_FAILED_ACTIVE_RD_NOT_SET`

Possible Cause This error code is returned when the active remote destination is not set.

Problem Error Message `CTIERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE`

Possible Cause This error code is returned when the enduser is not associated with the device.

Problem Error Message `CTIERR_DEVICE_ALREADY_REGISTERED_NONEXTEND`

Possible Cause This error code is returned when the device registration failed due to the device already being registered in non-extend mode.

Problem Error Message `CTIERR_MEDIA_ALREADY_TERMINATED_EXTEND`

Possible Cause This error code is returned when the device registration failed due to the device already being registered in extend mode.

Problem Error Message `CTIERR_INVALID_REMOTE_DESTINATION_NAME`

Possible Cause This error code is returned when an invalid remote destination name is entered.

CiscoProvAuthenticationInfoEv

CiscoProvAuthenticationInfoEv code returns to notify the application that the password is about to expire or has already expired. The application should have Provider Observer onto the Provider object to receive this event.

If the application invokes a connection and it fails because of an expired password, it will receive a PlatformException with a newly defined error code. For more information, see CiscoJtapiExceptions, on page 53.

In the case of a failover, the application will not explicitly request a connection, and will not receive a PlatformException. As the provider will already have an observer, it will deliver a CiscoProvAuthenticationInfoEv to it with getDaysUntilPasswordExpiry() = CiscoProvAuthenticationInfoEv.PASSWORD_EXPIRED.

CiscoRTPHandle Interface on Cisco RTP Events

The following interfaces are enhanced to allow applications to get a CiscoRTPHandle from the events:

- CiscoRTPInputStartedEv, on page 551
- CiscoRTPInputStoppedEv, on page 553
- CiscoRTPOutputStartedEv, on page 559
- CiscoRTPOutputStoppedEv, on page 569

CiscoRTPHandle represents the callID of the call in Cisco Unified Communications Manager and stays the same as long as the call is active on the terminal. At any particular terminal/address, although the call and the associated GCID can change, CiscoRTPHandle will remain constant.

Cisco Terminal Filter and ButtonPressedEvents

Prior to the JTAPI 2.0 release, Cisco Unified JTAPI applications did not have direct control over terminal events. Applications can now receive button pressed events by setting the appropriate filter in the terminal observer. Applications no longer need to add call observer to get RTP events.

When setButtonPressedEv gets enabled by using CiscoTermEvFilter, applications receive CiscoTermButtonPressedEv when a digit gets pressed on the phone.

The following new or changed interfaces exist for CiscoTerminal Filter and ButtonPressedEvents:

CiscoTerminal

void	`setFilter (CiscoTermEvFilter terminalEvFilter)` Allows an application to have more control over the events thatget delivered to the TerminalObserver.

CiscoTermEvFilter

boolean	`getButtonPressedEnabled()` Gets the enable or disable status of the button-pressed events for the terminal. The default value specifies disabled.

boolean	`getDeviceDataEnabled()` Gets the enable or disable status of the device data events for the terminal. The default value specifies disabled.
boolean	`getRTPEventsEnabled()` Gets the enable or disable status of the RTP events for the terminal. Thedefault value specifies disabled.
void	**`setButtonPressedEnabled (boolean enabled)`** Enables or disables the button pressed events for the terminal.
void	**`setDeviceDataEnabled (boolean enabled)`** Enables or disables the device data status events for the terminal.
void	`setRTPEventsEnabled (boolean enabled)` Enables or disables the RTP events for the terminal.

CiscoTermButtonPressedEv

int	`getButtonPressed ()`

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for CiscoTerminal Filter and ButtonPressedEvents, see Message Sequence Charts, on page 755

CiscoTermRegistrationfailed Event

This event gets provided to the application when CiscoMediaTerminal or CiscoRouteTerminal registration fails asynchronously. Usually when registration fails, the application gets a CiscoRegistrationFailedException; however, it is possible that the registration request was successful, but the CTI rejected the registration. This event is provided for the cases where the registration request is successful, but the registration gets rejected. The application should have TerminalObserver to receive this event. Upon receipt of this event, the applications should reregister with the new parameter, depending on the error code that is provided for this event.

The following list provides the errors that get returned and the actions to take, by the application, to resolve them.

Errors

Problem Error Message `CiscoTermRegistrationFailedEv.MEDIA_CAPABILITY_MISMATCH`

Possible Cause Registration cannot get done because the terminal is already registered. Do the second registration with the same media capability.

Solution Try re-registering with the same capability.

Problem Error Message `CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_NONE`

Possible Cause Registration cannot get done because the terminal is already registered with media termination type 'none'.

Solution Try re-registering with media termination type 'none'.

Problem Error Message `CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_STATIC`

Possible Cause Registration cannot get done because the terminal is already registered with static media termination. For static registration, the second registration is not allowed.

Solution Wait until the terminal UnRegisters.

Problem Error Message `CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_DYNAMIC`

Possible Cause Registration cannot get done because the terminal is already registered with dynamic media termination.

Solution Try re-registering with dynamic media termination.

Problem Error Message `CiscoTermRegistrationFailedEv.OWNER_NOT_ALIVE`

Possible Cause When trying to register the terminal, registration gets in a race condition.

Solution Try re-registering the terminal.

The following interface is defined for this event:

```
int getErrorCode () //
```

Returns the errorCode for this exception

No changes exist in the message flow.

Cius Persistency

Wireless devices introduced by Cisco, for example the Cisco Cius, have the capability to move between WiFi networks and still retain their registration to a single CiscoUCM. However, due to the change in the network the IP address of the device might undergo a change.To indicate this change in IP address of wireless devices like Cius, Cisco JTAPI will expose a new interface to applications with the 9.0.1 release.

The new provider event - CiscoProvTerminalIPAddressChangedEv, will indicate that the IP address of the terminal has changed. Applications may choose to ignore this new event if they do not plan to support a Cius device.

On receiving this event, applications can query for the changed IP address of the terminal using the methods exposed in the new event or on the CiscoTerminal. This interface will also expose the IP addressing mode of the terminal, based on which IPv4/IPv6 address of the terminal can be queried.

Sample Code

```
public synchronized void providerChangedEvent( ProvEv[] eventList )
{
  try
  {
    for ( int i  =  0; i < eventList.length; i++ )
    {
```

```
    case (eventList[i].getID()) ){
        switch:
CiscoProvTerminalIPAddressChangedEv.ID:
      Terminal term  =  eventList[i]
                        .getTerminal();
        int ipAddrMode  =  eventList[i].getIPAddressingMode();
        InetAddr ipV4Addr  =  null;
        InetAddr ipV6Addr  =  null;
        if(ipAddrMode  =  CiscoTerminal.IP_ADDRESSING_MODE_IPv4)
        {
          ipV4Addr  =  eventList[i].getIPv4Address();
        }
        else if(ipAddrMode  =  CiscoTerminal.IP_ADDRESSING_MODE_IPv6)
        {
          ipV6Addr  =  eventList[i].getIPV6Address();
        }
        System.out.println(" TerminalName  = " + term.getName()  +
           " ipAddressing Mode  = " + ipAddrMode +
           " IPv4 Address  = " + ipV4Addr +
           " IPv6 Address = " + ipV6Address);
      }
    }
    catch (exception e)
    {
      …
    }
  }
```

Interface Changes

See CiscoProvTerminalIPAddressChangedEv, on page 482 for more information.

Message Sequences

See Cius Persistency, on page 792.

Backward Compatibility

This feature is backward compatible.

Clear Calls

Cisco Unified JTAPI applications can clear phantom calls without dropping active calls. The CiscoAddress provides a clearCallConnections message to allow applications to clear the calls when no active calls exist on the Cisco Unified Communications Manager (formerly Cisco Unified Call Manager).

Click to Conference

Click to conference feature provides interfaces on SIP trunk for applications such as Instant Messenger (IM) to add parties to a conference. Users can add other parties to a conference or remove parties by using such applications. When click to conference is used to add a party to conference, a call is offered to the target address. Only one connection for target address is created on this initial call. This call then gets added to conference which results in a new callID for the call on the target address and connections for other addresses in the call are created on the new call.

This section describes the interface changes in Cisco Unified JTAPI to handle the interactions when an address is added to a conference by using click to conference feature. When click to conference feature is used, consult call does not occur and Cisco Unified JTAPI applications do not receive CiscoConferenceStartEv or CiscoConferenceEndEv.

The feature can be disabled by turning off the "ENABLE CLICK TO CONFERENCE" CallManager service parameter.

Interface Changes

CiscoFeatureReason, on page 402

Message Sequences

Click to Conference, on page 1086

Backward Compatibility

This feature is backward compatible. No change in Cisco Unified JTAPI applications when this feature is not configured or used.

Cluster Abstraction

The CTIManager provides a virtual representation of all the Cisco Unified Communications Managers in a cluster. Cisco Unified JTAPI applications communicate with the CTIManager instead of with a specific Cisco Unified Communications Managers. The CTIManager also maintains connection between Cisco Unified Communications Managers in a cluster. This allows a provider to represent any devices in the cluster under the CTIManager. Figure 4: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box, on page 59 illustrates "Figure 4: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box, on page 59." Figure 5: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client, on page 60 illustrates "Figure 5: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client, on page 60."

For more details about the cluster administration and device pool settings, refer to the Cisco Unified Communications Manager help information.

Figure 4: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box

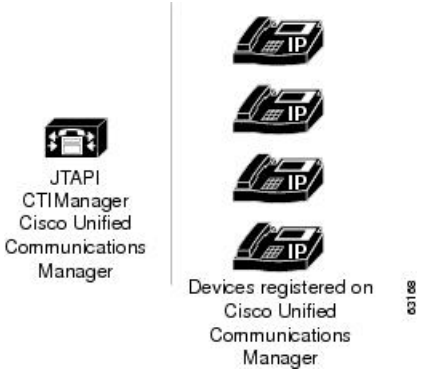
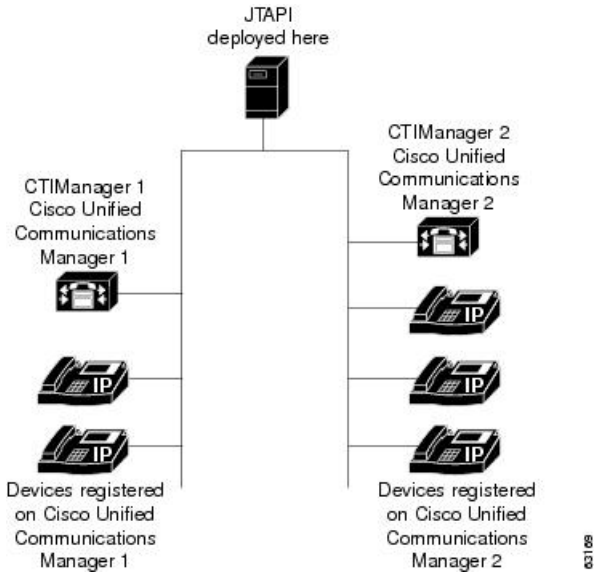


Figure 5: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client



Note In previous releases of Cisco Unified Communications Manager, applications that are running on Cisco Unified JTAPI could only control or monitor devices that are registered under a single Cisco Unified Communications Manager. If a Cisco Unified Communications Manager server went down, the connection between the Cisco Unified Communications Manager server and JTAPI would terminate and the Provider would shut down.

Command Line Invocation

This mode helps to install JTAPI in systems that do not have GUI support (for example, a Linux account). The entire installation procedure is guided by a character input based menu, where the user is asked to provide a series of inputs, based on the install time conditions. This mode also provides all the other options provided by the GUI based installer.

Component Updater

The Component Updater interface is enhanced to allow applications to specify the location of updater log. Currently the updater log is created in the same directory as the application. This enhancement allows applications to specify the trace location.

Interface Changes

See ComponentUpdater, on page 664

Message Sequences

See ComponentUpdater Enhancement Use Cases, on page 1230

Backward Compatibility

This feature is backward compatible.

Conference

When you conference two calls together, JTAPI specifies that all the parties from one call be moved to the other call. The call whose parties are moved away and that subsequently becomes invalid gets identified as the “merged” or “consult” call. The call to which the merged parties move gets identified as the “final” call hereafter. When parties move from the merged call to the final call, the application receives events that indicate that all parties dropped from the merged call and events that indicate that the parties reappeared on the final call.

To correlate the newly created connection objects with the old connection objects, use the CiscoConection.getConnectionID() method to obtain CiscoConnectionID objects for all old connections and all new connections. Matching connections will have identical CiscoConnectionID objects when you compare them by using the CiscoConnectionID.equals() method.

Conference support exists for the following methods:

- javax.telephony.callcontrol.CallControlCall.conference(Call)
- javax.telephony.callcontrol.CallControlCall.getConferenceController()
- javax.telephony.callcontrol.CallControlCall.getConferenceEnable()
- javax.telephony.callcontrol.CallControlCall.setConferenceController(TerminalConnection)
- javax.telephony.callcontrol.CallControlCall.setConferenceEnable(boolean)

Note As of Cisco Unified Communications Manager Release 8.6, Cisco TelePresence MCU conference bridges are supported through JTAPI/TSP. From a JTAPI/TSP perspective, this conference bridge behaves in the same way as other supported conference bridges.

Cisco Extensions

Cisco Unified JTAPI implementation provides two extra events that signal the Start and End of Conference: CiscoConferenceStartEv and CiscoConferenceEndEv. These events get sent when Conference initiates and when it completes. They give handles to the final call, the merged conference (consult) call, and the two controlling TerminalConnections (in HELD and TALKING state).

CiscoConferenceStartEv

This event gets sent when call1.conference(call2) is invoked or if the Conference button is pressed for the second time on an IPphone. The ConferenceStartEv signifies the start of the merging process. A sequence of merging events that are reflected by the Conference process in Cisco Unified Communications Manager follows.

CiscoConferenceEndEv

This event gets sent at the end of the merge process after a ConferenceStartEv is sent. It signifies the completion of the merge of the Consult (or Merged) call into the Final Conference Call. The Merged call specifies an INVALID state, and an ObservationEndedEv gets sent for the call observer.

CiscoCall.setConferenceEnable()

The Cisco Unified JTAPI implementation uses the CiscoCall.setConferenceEnable() and the CiscoCall.setTransferEnable() methods to control whether the consult call will be initiated via the conference or the transfer feature. If none of the features is enabled explicitly, transfer gets used by default.

Conference Scenarios

The following scenarios describe the two typical types of conference that can be invoked.

Consult Conference; B as the Conference Controller

The following sequence of steps typically describes this scenario:

- A calls B (Call 1).
- B answers.
- B Consults C (Call 2).

  setConferenceEnable()

  ```
  call2.consult(tc, C)
  ```

- C answers.
- B Completes Conference.

  ```
  Call1.conference(Call2)
  ```

Note You must invoke the conference() method on the original call to complete a conference after a consultation. Invoking conference in the consult call object throws an exception.

Arbitrary Conference; B as the Conference Controller

The following sequence of steps typically describe this scenario:

- A calls B (Call 1).
- B answers.
- B places the call on hold.
- B calls C (Call 2).
- C answers.
- B Completes Conference.

```
Call1.conference(Call2)
```
or

```
Call2.conference(Call1)
```

Conference Events

This table provides the sequence of Core, Call control, and Cisco Extension events when Call1.Conference(Call2) is called:

Table 1: Sequence of Events

Meta Event Cause	Call	Event	Fields
META_UNKNOWN	Call1	CiscoConferenceStartEv	consultCall = Call2finalCall = Call1conference Controller = TermConnB
META_CALL_MERGING	Call1	CallCtlTermConnTalkingEv B	
META_CALL_MERGING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_MERGING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_MERGING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv C CallInvalidEv C	consultCall = Call2finalCall = Call1conferenceController = TermConnB
META_UNKNOWN	Call2	CallObservationEndedEv	
META_UNKNOWN	Call1	CiscoConferenceEndEv	

Transfer and Conference Enhancement

All parties who are involved in the call transfer get sent CiscoTransferStartEv and CiscoTransferEndEv. All parties who are involved in the call conference get sent CiscoConferenceStartEv and CiscoConferenceEndEv. A call transfer still generates two events—the dropping of a connection to the first call and the creation of a connection to the second call. Cisco Unified Communications ManagerRelease3.1 changed this order of events. Connections first get created in the final call and then get dropped in the consult call.

Note In Cisco Unified Communications ManagerRelease3.0, not all parties who are involved in the transfer of calls received these events

Note Applications should not rely on the order of events between CiscoTransferStartEv and CiscoTransferEndEv or between CiscoConferenceStartEv and CiscoConferenceEndEv for transferring and conferencing when porting applications from Cisco Unified Communications ManagerRelease3.0 to 3.1.

Conference and Join

The Conference Feature provides the ability to conference more than two people into a single call. Events at CTI layer change, and Cisco Unified JTAPI gets enhanced to support the new CTI events.

Join Feature provides the ability to join multiple calls into one single conference call. This functionality now supports multiple calls. Applications need to pass an array of calls to be conferenced together.

The following new or changed interfaces exist for conference and joining of multiple calls into one conference call:

- The following interface allows Join to conference multiple calls into one conference call:
  `Call.Conference(Call[] otherCalls)`

Note A precondition requires that all the otherCalls must have controller as one leg of the call.

- The following new or modified interfaces exist in CiscoConferenceStartEv:
  - TerminalConnection getHeldConferenceController()—This interface proves useful only for the arbitrary conferencing of two calls and returns only one of the held calls.
  - TerminalConnection[] getHeldConferenceControllers()—This interface gets all of the held calls when multiple calls are joined.
  - TerminalConnection getTalkingConferenceController()—This interface returns the talking conference controller; however, if no talking conference controller exists when all the calls that are being joined into conference are held, this interface returns null.
  - Call getConferencedCall()—This interface returns only one of the many calls that are going to join into a conference and may not have any meaning for a join conference when more than two calls exist.

- New interface in CiscoConferenceEnded event Boolean isSuccess():

  This interface returns true or false depending on whether conference is successful or failed. Application can use interface to find whether conference is successful. The following events get defined as conference failure:

- If the application issues the request Call.conference(otherCalls[]), this conference would be considered failed if one or more than one calls could join into conference. Applications can use the interface getFailedCalls() to find the failed call.
- If no conference bridge is available and the conference could not complete at all, the application can use getFailedCalls() to get a list of calls that could not join the conference.
- A party that was being conferenced dropped out before conference could complete.

- An interface on the CiscoConferenceEnded event (Call[] getFailedCalls()) gets all the calls that failed to join the conference when the conference fails.

The following new or changed behaviors exist for Conference:

- No hold or unHold message such as applications see when an arbitrary conference occurs.
- An arbitrary conference does not require, as a precondition, that any calls be in a talking state; however, all the otherCalls must have a controller as one leg of the call.
- Applications can conference two or more held calls into a conference call. In finalCall, the controller automatically gets retrieved to a talking state.
- Always include an active call in the request Call.Conference(otherCalls). If an active call is not included in the conference request, the request fails.
- If no active call exists at the controller, the Call.Conference(otherCalls) request remains successful; however, if one active call exists, it the request must include it.
- If the application does not have an active TerminalConnection that is passed as an argument, Call.consult() throws a PreConditionException/InvalidArgumentException.
- If the controller does not have an active TerminalConnection, Call.Conference()/Call.Conference(Call[]) throws a PreconditionException/InvalidArgumentException.

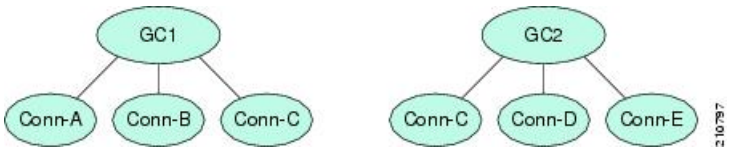
For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for conference and join, see Message Sequence Charts, on page 755

Conference Chaining

The conference chaining feature lets applications join two separate conference calls together. JTAPI applications see chained conference calls that are represented as two separate calls. When conference calls are chained, JTAPI creates a new connection for the conference chain and provides the CiscoConferenceChainAddedEv event on CallCtlCallObserver. When the conference chain is removed from the call, JTAPI disconnects the conference chain connection and provides the CiscoConferenceChainRemovedEv event on CallCtlCallObserver. From CiscoConferenceChainAdded/RemovedEv, applications can obtain CiscoConferenceChain, which provides a link for all the conference chain connections.

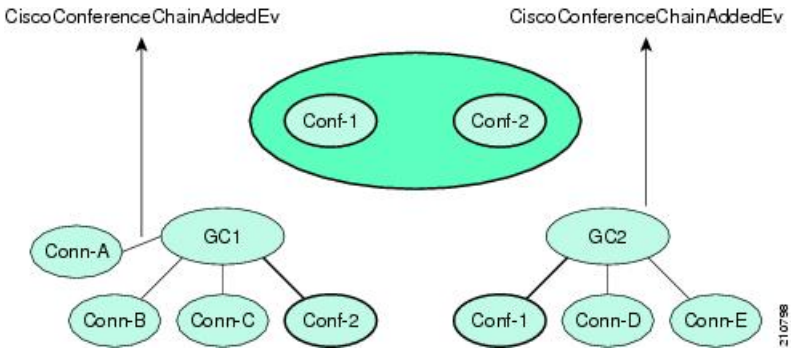
The following figure shows parties A, B, and C in conference call GC1 and parties C, D, and E in conference call GC2.

Figure 6: Calls Prior to Chaining



After the conference chain is created, the calls will look like the following figure.

Figure 7: Calls After Chaining



Applications may get all the participants of a chained conference from the CiscoChainedConference object, which provides conference chain connections from all the conference calls that are chained together. By browsing through the connections list, applications can get a list of all the chained conference calls; however, applications must have at least one participant of each conference that is observed.

Note For any conference scenario that involves chaining conferences or adding parties to a chained conference call, JTAPI will not provide ConferenceStarted/Ended event.

For more information, see the following topics:

- CiscoCall, on page 326 (for the getConferenceChain() interface)
- CiscoConferenceChain, on page 365
- CiscoConferenceChainAddedEv, on page 366
- CiscoConferenceChainRemovedEv, on page 369

Consult Without Media

Applications can inform Cisco Unified Communications Manager that a consultation call for a transfer is being placed without establishing the media path. The system does not require establishing the media path for the intermediate call, if the consultation call is being placed to determine whether an agent is available before the actual transfer. The consultWithoutMedia method as defined in the CiscoConsultCall interface creates a consultation call without establishing the media path.

Note The system allows only transferring of the consultation call; it does not allow the call to be in conference.

CTI Ports

CTI Ports that are registered by an application include a mechanism similar to phone devices. When the Cisco Unified Communications Manager that is handling signaling for a CTIPort fails, the CTIManager recovers its services according to the device pool administration for this device. On a CTIManager failure, Cisco Unified JTAPI reregisters the CTI Port after it connects to the backup CTIManager. The CiscoAddrOutOfServiceEv and CiscoTermOutOfServiceEv events and the corresponding in-service events get sent after recovery of the CTI Port.

The application controls media streaming for these devices, and streaming continues even when the port is out of service. The application has responsibility to ensure that new calls do not get presented to the device until it is ready to accept them.

CTI RoutePoints

On a Cisco Unified Communications Manager server failure, the CTIManager recovers the device from the Cisco Unified Communications Manager server group as defined in the device pool administration for the CTI RoutePoint. When the primary Cisco Unified Communications Manager server recovers, the CTIManager attempts to recover the device on its primary Cisco Unified Communications Manager. This re-homing happens when no more calls exist on the device (similar to physical devices).

On a CTIManager failure, Cisco Unified JTAPI recovers the device on the backup CTIManager. The application receives notification of the availability of a device with the CiscoAddrOutOfServiceEv and CiscoAddrInServiceEv events.

CTI Remote Device for JTAPI

Changes in personal device preferences and an increasing number of mobile and remote workers necessitates a more flexible Unified Communications solution that extends UC features with a Bring Your Own Device philosophy. Extend and Connect addresses this change.

Extend and Connect is a feature that allows administrators to rapidly deploy Unified Communications (UC) Computer Telephony Integration (CTI) applications which interoperate with any endpoint. Extend and Connect lets users leverage the benefits of UC applications from any location using any device. This feature allows interoperability between newer UC solutions and legacy systems, so customers can migrate over time as existing hardware is deprecated.

For more information, please refer to the *Cisco Unified Communications Manager Features & Services guide*.

Interface Changes

See CiscoRemoteDestinationInfo, on page 507, CiscoProvTerminalRemoteDestinationChangedEv, on page 505, CiscoProvider, on page 486, CiscoTerminalProtocol, on page 637.

Message Sequences

See CTI Remote Device, on page 801.

Backward Compatibility

This feature is backward compatible.

Play Announcement

Play Announcement allows a specified preconfigured announcement to be played or streamed to a remote destination. Only announcements that are uploaded to the Cisco Unified Communications Manager can be played. All announcement requirements and limitations are applicable to Play Announcement. As part of this feature, new JTAPI APIs, events, and error codes are added.

Only CTI Remote Devices with a persistent call support play announcement. Play announcement is not supported on IP phones or CTI ports. Cisco recommends that the persistent call plays an announcement when answered. Announcements can be played on persistent calls even without customer calls. However, if there are customer calls incoming to the remote device, announcements are played only when that call is not in a connected state. Multiple announcements cannot be played at the same time. No features (transfer, conference, hold) can be performed on the announcement call.

The following are required for the application to play the announcement: at least one remote destination must be configured, the active remote destination must be set, and a persistent call must be created.

JTAPI supports a new API, CiscoCall.startAnnouncement(), which allows applications to start to play an announcement. This API creates an announcement call. This newly created announcement call counts toward both the busy trigger and maximum calls limit. JTAPI APIs such as Provider.getCalls(), Address.getConnections(), and Terminal.getTerminalConnections() return information for the announcement call.

No new APIs are added to disconnect/drop the announcement calls. Use Existing Call.drop() and Connection.disconnect() JTAPI APIs to disconnect the announcement calls. In addition to the APIs that explicitly end the announcement call, the announcement call is also automatically disconnected after the announcement is complete. Any state change in the announcement call stops the announcement, and also disconnect the announcement call. For example, if there is an incoming customer call in ringing state and the announcement is still being played, after the customer call is answered, the announcement call is disconnected.

As a part of this feature, new JTAPI events are introduced. The CiscoAnnouncementStartedEv is a new JTAPI event that is delivered to applications, notifying applications when the play announcement starts. To notify applications when the play announcement has ended, another new JTAPI event, the CiscoAnnouncementEndedEv, is delivered to apps. If during any time, an error occurs during play announcement, a new JTAPI event delivers that information to apps as well: CiscoAnnouncementErrorEv.

Some of the new JTAPI error codes that are introduced as part of this feature include:

- CiscoJtapiException.CTIERR_NO_PERSISTENT_CALL_EXISTS: This error codes indicates that no persistent call exists.
- iscoJtapiException.CTIERR_ANNOUNCEMENT_ALREADY_IN_PROGRESS: This error code indicates that there is already an announcement in progress.
- CiscoJtapiException.CTIERR_ERROR_PLAYING_ANNOUNCEMENT: This error code indicates that there is an error in playing the announcement.
- CiscoJtapiException.CTIERR_PLAY_ANNOUNCEMENT_FAILED: This error code indicates that play announcement failed.

Interface Changes

- CiscoAddress, on page 283
- CiscoAnnouncementStartedEv, on page 322
- CiscoAnnouncementEndedEv, on page 322
- CiscoAnnouncementErrorEv, on page 323
- CiscoFeatureReason, on page 402

Message Sequences

See Play Announcement, on page 1350.

Backward Compatibility

This feature is backward compatible and existing applications are not affected by its introduction.

Verify Remote Destination Support

In Cisco Unified Communications Manager 10.0(1), the existing CiscoRemoteTerminal.addRemoteDestionation(), CiscoRemoteTerminal.updateRemoteDestination(), and CiscoRemoteTerminal.updateRemoteDestinationNumber() APIs are enhanced to allow validation of the remote destination. As part of this feature, when an application attempts to add or update a remote destination using JTAPI API, Cisco JTAPI validates the remote destination to determine whether the destination is reachable. If the destination is not reachable, the add or update remote destination request returns an error of CiscoJtapiException.CTIERR_EXTEND_AND_CONNECT_DESTINATION_NOT_REACHABLE. The remote destination is then not updated in the database. A sucessful update is possible only if the remote destination is reachable, and the database is then updated with the remote destination number. The verification of the remote destination in the update applies only when the JTAPI API is invoked. Adding or updating the remote destination information through the ccmadmin page does not result in the verification of the remote destination. No new APIs are added as part of this feature. A new error is introduced.

Interface Changes

CiscoJtapiException, on page 410

Message Sequences

Verify Remote Destination Support, on page 1446

Backward Compatibility

This feature is backward compatible and existing applications are not affected by this enhancement.

NuRD (Number Matching for Remote Destination) Support

Starting in Cisco Unified Communications Manager 10.0(1), the existing “Cisco Extend and Connect” feature is enhanced to include number matching for remote destination support. When users make a direct call to a number that is configured as a remote destination for CTI Remote Device (CTI RD), and if that remote destination is active, the call is offered on the CTI Remote Device and extended to the remote destination. From the application, the current called party is the CTI RD. If the active remote destination is not set, when users call a remote destination number, the call will be a direct call between the caller and the remote destination.

The same situation applies to a call from a remote destination to an enterprise dial number. If the remote destination is active, the CTI RD is initiating the call to the enterprise dial number. If the active remote destination is not set, calls from a remote destination to an enterprise dial number are direct calls between the remote destination and the enterprise dial number.

For those calls from and to a remote destination number, all existing features that are allowed on CTI RD are available.

Interface Changes

There are no interface changes for this feature.

Usage Cases

Use Cases for NuRD (Number Matching for Remote Destination), on page 1301

Backward Compatibility

This feature may change existing expected behavior in direct calls to and from remote destination numbers. Applications that do not leverage this NuRD feature keep the clusterwide service parameter “Reroute Remote Destination Calls to Enterprise Number” set to False. Enabling the parameter enables the NuRD features. This parameter is set to False by default.

Mobility Interaction Support

Starting in Cisco Unified Communications Manager 10.0(1), the existing “Cisco Extend and Connect” feature is extended to include mobility interaction. Users can now specify remote destinations that are shared between the CTI Remote Device (CTI RD) and the Remote Destination Profile (RDP). When both the CTI RD and the RDP are configured for the same user, and if the application is active (active rd is set), CTI RD will process the call first and then offer the call to the RDP. If the application is not active, the RDP processes the call first and does not offer the call to the CTI RD. When only CTI RD is configured for a user, the existing "Cisco Extend and Connect" feature behavior with remote destinations remains unchanged. When only RDP is configured for a user, there is no application support because the devices are not CTI controllable.

Interface Changes

There are no interface changes for this feature.

Usage Cases

Mobility Interaction Support, on page 1259

Backward Compatibility

This feature is backward compatible and existing applications are not affected by the enhancement.

CTI RD Call Forward

Beginning in Release 10.0(1), CTI RD Call Forward provides applications with the ability to control when incoming calls are forwarded to all configured Remote Destinations on the CTI Remote Device, when no active remote destination is set.

A new check box **Route calls to all remote destinations when client is not connected**, is added to the Cisco Unified Communications Manager device page. The check box determines whether calls are routed to all remote destinations when active remote destination is not set.

When the check box, **Route calls to all remote destinations when client is not connected** is enabled, and Active Remote Destination is not set, the call is routed to all remote destinations. If this check box is disabled, and Active Remote Destination is not set, the call will be disconnected with User_Busy error.

In scenarios where Active Remote Destination is set, the call will be always routed to the Active Remote Destination even if the check box **Route calls to all remote destinations when client is not connected** is selected.

Interface Changes

There are no interface changes for this feature.

Use Cases

CTI RD Call Forward, on page 870

Backward Compatibility

Applications should enable the check box **Route calls to all remote destinations when client is not connected** to maintain the old behavior.

CTI Video Support

The CTI Video Support feature allows the JTAPI Application to detect the multimedia capabilities of Line Devices; such as receiving video, sending video and both receiving and sending video. Cisco JTAPI provides the applications with the ability to expose the video capabilities of a terminal through the enhancement CTI Video Support. CTI applications can differentiate a video-enabled device from a non video-enabled device, and, a video call from an audio only call.

Cisco JTAPI provides a new API, getCiscoMultiMediaCapabilityInfo() on the CiscoTerminal to expose the multimedia capabilities of the device. Cisco JTAPI exposes the multimedia capabilities of the terminal after the device is in registered state. The multimedia capabilities of the terminal include:

- video capability (either none or video enabled),
- telepresence interoperability (either none or telepresence interoperability enabled on the device), and,
- screen count (to know the number of screens available on device).

The multimedia capabilities are exposed on a new interface CiscoMultiMediaCapabilityInfo, which has the following APIs to expose these capabilities.

- getVideoCapability(),
- getTelepresenceInfo(), and,
- getScreenCount().

The following APIs on the CiscoCall are used by the application to determine the calling party or called party multimedia capability prior to media setup.

- getCallingTerminalMultiMediaCapabilityInfo()—of the calling party in a call
- getCalledTerminalMultiMediaCapabilityInfo()—of the called party in a call

When the video capability of the terminal changes, a new Cisco JTAPI event, CiscoProvTerminalMultiMediaCapabilityChangedEv, notifies the application. This event is a JTAPI provider event, and is delivered when the application adds a Provider Observer. The terminal must be in registered state, to receive this event. Plugging in or plugging out the Cisco camera will not affect the video capability status, therefore, the event will not be triggered. However, you can modify the video capability using the Cisco UCM Administration Interface > Device Configuration page.

Note The initial video capability API on CiscoTerminal is not supported for CTI Route Points and CTI Ports; however, they can receive the video information of the calling party.

The following devices supports the CTI Video feature:

- 89xx (SIP only)
- 99xx
- E20
- EX60/90
- CTS 500
- CTS 500-32
- Jabber(CSF)
- CTI RoutePoint
- CTI Port

Interface Changes

See the following sections for interface changes:

- CiscoCall, on page 326
- CiscoMultiMediaCapabilityInfo, on page 462
- CiscoProvTerminalMultiMediaCapabilityChangedEv, on page 483
- CiscoTerminal, on page 611

Message Sequences

See CTI Video Support, on page 879.

Backward Compatibility

This feature is backward compatible.

Default CTI IP Addressing for Devices

A new CTIManager service parameter, **IP Addressing Mode for Devices**, has been added that allows you to configure the default CTI IP addressing mode for a device that does not have an associated Common Device Configuration.

When an application invokes the CiscoTerminal.getIPAddressingMode() method for a device that does not have a Common Device Configuration, JTAPI returns the value of the service parameter. The default setting for the new service parameter is **IPv4 and IPv6**. JTAPI communicates the value via CiscoTerminal.IP_ADDRESSING_MODE_IPV4_V6.

Note For an individual CTI device, if that device has an associated Common Device Configuration, the IP Addressing Mode setting in the Common Device Configuration overrides the value of the **IP Addressing Mode for Devices** service parameter.

DeleteCall

DeleteCall interface provides applications with the ability to delete a call that was created by using the createCall interface. This method accepts a call and throws an InvalidStateException if a provider is not in service or if the call is not in the IDLE state. DeleteCall moves the call to the INVALID state.

The following interface gets added to **`CiscoProvider`**:

```
{ public void deleteCall( Call call ) throws InvalidStateException;
}
```

Applications can use this interface to delete the call that was created by using createCall interface. This method accepts a call and throws an InvalidStateException if the provider is not in service or if the call is not in the IDLE state. DeleteCall moves the call to the INVALID state.

To successfully delete a call, the application creates the call by using createCall, and the call should be in the IDLE state.

Device Recovery

Cisco Unified JTAPI supports automatic device recovery.

Device Recovery for Phones

For devices such as the Cisco Unified IPPhone 7960, the re-homing feature represents part of the device firmware. On a primary Cisco Unified Communications Manager failure, the phone attempts to connect to the backup Cisco Unified Communications Manager when it is no longer on a call. This transition gets communicated to applications in the form of out-of-service and in-service events described in CTIManager Failure, on page 149.

For virtual devices with no firmware such as CTI Ports and CTI RoutePoints, the CTIManager or Cisco Unified JTAPI performs the failover.

Device State Server

The Device State server provides the cumulative state of all the addresses on a terminal. These events are delivered as TerminalEvent. Applications need to add TerminalObserver to get these events.

The states follow:

- IDLE—If no calls exist on any of the addresses on the terminal, consider the DeviceState as IDLE, and Cisco Unified JTAPI sends CiscoTermDeviceStateIdleEv to applications.
- ACTIVE—If any addresses on the terminal have an outgoing call (in CTI State Dialtone, Dialing, Proceeding, Ringback, or Connected) or an incoming call (in CTI State Connected), the consider DeviceState as ACTIVE, and Cisco Unified JTAPI sends CiscoTermDeviceStateActiveEv to the application.
- ALERTING—If address on the terminal has an outgoing call (in CTI State Dialtone, Dialing, Proceeding, Ringback, or Connected) or an incoming call (in CTI State Connected) and at least one of the addresses on the terminal has an unanswered incoming call (in CTI State Offering, Accepted, or Tinging), the DeviceState is ALERTING, and Cisco Unified JTAPI sends CiscoTermDeviceStateAlertingEv to the application.
- HELD—If all the calls on any of the address on the terminal are held (in CTI State OnHold), the DeviceState is HELD and Cisco Unified JTAPI sends CiscoTermDeviceStateHeldEv to the application.

New Events

- CiscoTermDeviceDeviceStateIdleEv
- CiscoTermDeviceStateActiveEv
- CiscoTermDeviceStateAlertingEv
- CiscoTermDeviceStateHeldEv

New and Changed Interfaces

public int getDeviceState() returns the device state of the terminal.

The following new interfaces on CiscoTermEvFilter set and get the device state:

void	`setDevideStateActiveEvFilter(boolean filterValue)` Enables and disables the CiscoTermDeviceStateActiveEv filter.The default value is disable.
void	`setDeviceStateAlertingEvFilter(boolean filterValue)` Enables and disables the CiscoTermDeviceAlertingEv filter.Thedefault value is disable.
void	`setDeviceStateHeldEvFilter(boolean filterValue)` Enables and disables the CiscoTermDeviceHeldEv filter.Thedefault value is disable.

void	`setDeviceStateIdleEvFilter(boolean filterValue)` Enables and disables the CiscoTermDeviceIdleEv filter. Thedefault value is disable.
boolean	`getDeviceStateActiveEvFilter()` Gets the CiscoTermDeviceStateActiveEv filter status.
boolean	`getDeviceStateAlertingEvFilter()` Gets the CiscoTermDeviceStateAlertingEv filter status.
boolean	`getDeviceStateActiveEvFilter()` Gets the CiscoTermDeviceStateAlertingEv filter status.
boolean	`getDeviceStateActiveEvFilter()` Gets the CiscoTermDeviceStateAlertingEv filter status.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243

Direct Transfer Across Lines

The Direct Transfer Across Lines feature allows support for connected transfer across lines. It allows two calls on different addresses of the same terminal to be transferred though the Transfer softkey on the phone or by using the transfer() API that is provided by Cisco Unified JTAPI. When a transfer is performed across lines, the JTAPI application behavior changes, as applications do not see a common controller address in final and consult calls. There is no change in the API and the same events are delivered whether calls are transferred on the same address (regular transfer) or across addresses (direct transfer across lines). This feature is supported on all supported phones, including CTI port, SCCP devices and SIP devices.

If an observer is not added to either of the two addresses from which the direct transfer is being attempted from the JTAPI API, then Cisco Unified JTAPI throws PlatformException with this error: Transfer controller is not set and could not find a suitable TerminalConnection.

Usage Guidelines

The points below indicate how applications must use the Direct Transfer Across Lines feature:

- Applications must add Call Observer on the both the lines across which they try a direct transfer.
- Earlier, applications were recommended to check if both the calls have a common address and if that address is on the same Terminal. For Direct Transaction Across Lines, it is not required to check this, if the address is common between two calls across which direct transaction is invoked. It must be ensured that both the calls should each have an address which exists in a common terminal.
- Cisco Unified JTAPI reports the same set of events, as it does currently, for transferring of a call on same address. Applications are not required do anything with these calls after invoking Transfer() until receiving CiscoTransferEndEv.
- As transfer is done across addresses, applications do not get a common controller in CiscoTransferStartEv and should upgrade the application logic.

Event Flow Comparison and Sample Code

The following table provides details of the event flow.

Table 2: Event Flow Comparison and Sample Code for Transfer Invocation

Transfer on Same Lines	Transfer Across Lines
Setup	
Address A on Terminal T1 Address B1, B2 on Terminal T2 Address C on Terminal T3	Address A on Terminal T1 Address B1, B2 on Terminal T2 Address C on Terminal T3
Feature Invocation	
A calls B1 [GC1 = GolbalCallID1] GC1: Connection A1-> Conn1 GC1: Connection B1->Conn2 B1 calls C [GC2 = GolbalCallID2] GG2: Connection B1-> Conn3 GC2: Connection C->Conn4 GC1.transfer(GC2);	A calls B1 [GC1 = GolbalCallID1] GC1: Connection A->Conn1 GC1: Connection B1->Conn2 B2 calls C [GC2 = GolbalCallID2] GG2: Connection B2->Conn3 GC2: Connection C->Conn4 GC1.transfer(GC2);
Events Delivered to Application (assuming all parties are observed)	

Transfer on Same Lines	Transfer Across Lines
GC1:	GC1:
CiscoTransferStartEv	CiscoTransferStartEv
[getTransferControllerAddress() returns B1]	[getTransferControllerAddress() returns B1]
ConnCreatedEv for C	ConnCreatedEv for C
ConnConnectedEv for C	ConnConnectedEv for C
CallCtlConnEstablishedEv for C	CallCtlConnEstablishedEv for C
TermConnCreatedEv for T3(Address C)	TermConnCreatedEv for T3(Address C)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B1
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B1
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B1)
CallCtlTermConnDroppedEv for T2(Address B1)	CallCtlTermConnDroppedEv for T2(Address B1)
CiscoTransferEndEv	CiscoTransferEndEv
GC2:	GC2:
CiscoTransferStartEv	CiscoTransferStartEv
[getTransfeControllerAddress() returns B1]	[getTransferControllerAddress() returns B1]
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B2)
CallCtlTermConnDroppedEv for T2(Addresss B1)	CallCtlTermConnDroppedEv for T2(Addresss B2)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B2
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B2
TermConnDroppedEv for T3(Address C)	TermConnDroppedEv for T3(Address C)
ConnDisconnectedEv for C	ConnDisconnectedEv for C
CallCtlConnDisconnectedEv for C	CallCtlConnDisconnectedEv for C
CallCtlTermConnDroppedEv for T3(Address C)	CallCtlTermConnDroppedEv for T3(Address C)
CiscoTransferEndEv	CiscoTransferEndEv
CallInvalidEv	CallInvalidEv
CallObservationEndedEv	CallObservationEndedEv
Note GC2 - Disconnect events are for Address B1 on Terminal T2	**Note** GC2 - Disconnect events are for Address B2 on Terminal T2

Note In connected Transfer Across Lines scenario, apart from events mentioned, applications can see another temporary call GC3 going active(CallActiveEv) and GC3 goes idle (CallInvalidEv) immediately after the transfer is completed

Transfer on Same Lines Sample Code

```
Handle(CiscoCallEv event)
{
  …..
  …..
  if (event instanceOf  CiscoTransferStartEv)
  {
    CiscoTransferStartEv ev  =
      (CiscoTransferStartEv)event;
    processTransfer(ev);
  }
}
processTransfer(CiscoTransferStartEv ev){
  CiscoAddress commonAddr  =
    ev.getTransferControllerAddress();

  CiscoCall GC2  =  ev.getTransferringCall();
  CiscoCall GC1  =  ev.getFinalCall();
  CiscoConnection droppedConn1  =
    findConnection(GC1,  controllerAddr);

  CiscoConnection droppedConn2  =
    findConnection(GC2,  controllerAddr);
  //Additional App logic to clear connections.
}
Connection findConnection(CiscoCall GCx,  CiscoAddress addr){
  CiscoConnection[] conns  =  GCx.getConnections();
  for (i = 0; i<conns.length; i++)
  {
    if conns[i]
      .getAddress().equals(addr) {
      return conns[i];
    }
  }
}
```

Note Application logic is based on common transferControllerAddress and works fine in this case, because commonAddr is there in both final and consult call

Transfer Across Lines Sample Code

```
Handle(CiscoCallEv event)
{
  …..
  …..
  if (event instanceOf  CiscoTransferStartEv)
  {
    CiscoTransferStartEv ev  =
      (CiscoTransferStartEv)event;
    processTransfer(ev);
  }
}
processTransfer(CiscoTransferStartEv ev){
  String termName  =  ev.getControllerTerminalName();

  CiscoCall GC2  =  ev.getTransferringCall();
  CiscoCall GC1  =  ev.getFinalCall();
  CiscoConnection droppedConn1  =  findConnection(GC1,  termName);
```

```
  CiscoConnection droppedConn2  =  findConnection(GC2,  termName);
  //Additional App logic to clear connections.
}
Connection findConnection(CiscoCall GCx,  String termName){
  CiscoConnection[] conns  =  GCx.getConnections();
  for (i = 0; i<conns.length; i++)
  {
    CiscoTerminalConnection[] termConns  =
      conns[i].getTerminalConnections();
    for(j = 0; j<termConns.length; j++)
    {
      if(termConns[j].getTerminal().getName.equals(termName)
          && termConns[i].getState() ! =
          TerminalConnection.PASSIVE)
      {
        return termConns[i].getConnection();
      }
    }
  }
}
```

Note There is no common address for controllers in final and consult call, but the controller TerminalName is same for both the controller Addreses. So, application should rely on CommonTerminalName to find out the connections, terminal connections and controllers.

Interface Changes

See CiscoTransferStartEv, on page 659

Message Sequences

See Direct Transfer Across Lines Use Cases, on page 1151

Backward Compatibility

This feature is backward compatible. To provide backward compatibility for applications, a new permission to devices that allow connected transfer across lines has been added, along with a new standard role and a standard user group for this permission. Applications can control these devices only if this new role Standard Supports Connected Xfer/Conf is associated to the application user. Applications will be able to control these devices only if this new role "Standard CTI Allow Control of Phones supporting Connected Xfer/Conf" is associated to the application user. So, by default these devices are listed as restricted, assuming that the application uses JTAPI 7.1.2 or higher and only if application upgrades to handle this feature and associates the new permission can it control these devices. If the application uses an older JTAPI client the devices are not restricted but if the application tries to observe these devices (which supports this feature to be invoked manually), JTAPI throws an exception and marks these devices as restricted from there on.

However, the application can invoke DirectTransfer Across Lines from existing JTAPI transfer() API on any type of phone and there is no restriction on this behavior as applications are expected to issue this request only if they support this feature. Also, a FarEnd point performing a Direct/Connected Transfer Across Lines is uncontrolled and can cause problems to applications. This means that JTAPI always reports events for Direct Transfer Across Lines for all the phones.

Be aware that any old JTAPI application will not have any BWC issues if it is run in an environment where Direct Transfer Across Lines is not invoked (either on phones or through JTAPI API). However, applications changes are required if this this feature is used in such a setup.

Cisco assumes that two or more applications do not control or observe the same terminal or address simultaneously. If they do, all instances of this application make changes to support this feature or coordinate to avoid any problem. Otherwise, application behavior may be unforeseen. For example, if App1 and App2 are two applications controlling or observing the same terminal or address and App1 makes changes to support this feature then App2 is also expected make changes to support the feature. Else, invocation of this feature by App1 on common devices can break App2.

As, the feature is designed to provide an enhanced user experience, Cisco strongly recommends that all Cisco Unified JTAPI applications should evaluate and support this feature and upgrade if necessary with the code logic to handle both the old and new behavior.

Directed Call Park

This feature allows the user to park a call by transferring the call to a user-selected park code.

Examples

A calls B; B transfers the call to a parked DN. On completion of the transfer, the A to B call is parked at the specified parked DN. A will receive MOH (if configured). When C unparks the call (by dialing the prefix code and park code), A and C connect.

If A calls the parked DN directly, A connects to the parked DN, and the system marks this parked DN as busy. A stays connected to this parked DN until park reversion.

If C does not unpark the call at the parked DN, the call park reverses to the DN that parked the call (B), and A and B connect again. B can again try to d-Park to another parked DN. When park reversion occurs, Cisco Unified Communications Manager JTAPI passes a new reason code to the application.

CTI sends the parked number to Cisco Unified Communications Manager JTAPI in the form “Park Number: (<Prefix Code>)<DPark DN>”. Cisco Unified Communications Manager JTAPI parses this and exposes both Prefix Code and DPark DN to applications.

When a call is unparked, the parked party and unparking party both receive a CPIC event with the reason given by CTI, and the parked party connects to the unparking party.

When party A calls a dPark DN and party B also calls the same dPark DN, the system can connect either A or B to the dPark DN, and the other party is disconnected.

Cisco Unified Communications Manager JTAPI Support

Cisco Unified Communications Manager JTAPI supports this feature. When the system transfers a call to a directed call park DN (dparked), the application sees a connection created for directed call park DN, and the call control connection state is CallControlConnection.QUEUED. The system delivers CiscoTransferstart and end events. An application can use the new interface on CiscoConnection to get the prefix code needed to unpark the call.

Performance and Scalability

This feature provides the same performance impact as the existing transfer feature.

Directory Change Notification

Applications require notification asynchronously of device additions or deletions from the user control list and device deletions from the Cisco Unified Communications Manager database. Applications also receive notification about line changes to a device. This notification gets sent to Cisco Unified JTAPI and propagates to applications with CiscoAddrCreatedEv, CiscoAddrRemovedEv, CiscoTermCreatedEv, and CiscoTermRemovedEv on the AddressObserver and TerminalObservers, respectively.

Note Ensure that the device is registered for CTIPorts and CTIRoutePoints to receive the line change notification.

Do Not Disturb

Do-Not-Disturb (DND) gives phone users the ability to go into the DND state on the phone when they are away from their phones or do not want to answer the incoming calls. The DND softkey enables and disables this feature.

From the user windows, users can configure the following settings for DND:

- DND Option-Ringer off
- DND Incoming Call Alert-beep only/flash only/disable
- DND Timer-value between 0-120 minutes. It specifies a “period in minutes to remind the user that DND is active”.
- DND status-on/off

Note The Application can only enable or disable the DND status.

- The Application can set the DND status by invoking a new interface on CiscoTerminal.
- JTAPI will also query the application about any change in the DND status when DND status is set by phone, Cisco Unified Communications Manager Administration, or application.
- The application must enable the filter in CiscoTermEvFilter to receive the preceding notification.
- The application can also query for the DND status through a new interface on CiscoTerminal.
- The application can also query for the DND option through a new interface on CiscoTerminal.

Note This feature applies to phones and CTI ports. It does not apply to Route points.

In the case of emergency calls (made by a CER application) landing on an application that has DND enabled, the system overrides the DND settings and presents the call to the application. A new parameter, FeaturePriority, in the redirect() and selectRoute() APIs on CiscoCall, CiscoConnection, and CiscoRouteSession, respectively,

makes this possible. The CER application that initiates the emergency call sets FeaturePriority as FeaturePriority_Emergency. The application sets the feature priority only for emergency calls. In the case of normal calls, applications either do not set the feature priority at all or set it to FeaturePriority_Normal. Applications do not set FeaturePriority_Emergency in case of normal calls. When initiating feature calls such as intercom, applications must specify FEATUREPRIORITY_URGENT.

Note The connect() API on CiscoCall does not support the FeaturePriority parameter.

The application receives an exception if it tries to perform a getDNDStatus(), setDNDStatus(), or getDNDOption() before the device is in service.

A Post condition is added to DND to handle a DB update failure or device out-of-service situations if they occur after the setDNDStatus() request is sent. If a DB update failure or device out-of-service condition occurs after the setDNDStatus() request is sent, setDNDStatus() delivers a CiscoTermDNDStatusChangedEv to the application. If this event is not received, a post-condition time-out occurs, and the following exception is thrown: could not meet post conditions of setDNDStatus().

Backward Compatibility

This feature is backward compatible. Applications recognize new events if this feature is configured. You can filter the new events through the TerminalEventFilter interface (CiscoTermEvFilter). By default, this filter is disabled and the system does not deliver the new events.

For additional information, see the following topics:

- CiscoTerminal, on page 611
- CiscoTermDNDStatusChangedEv, on page 604
- CiscoTermEvFilter, on page 608
- CiscoCall, on page 326
- CiscoConnection, on page 380
- CiscoRouteSession, on page 517
- CiscoTermInServiceEv, on page 638

Do Not Disturb-Reject

Do Not Disturb–Reject (DND–R) is an enhancement to the existing DND feature. Cisco Unified Communications Manager and JTAPI previously supported only the Ringer off DND. The user can reject calls with DND–Reject. You can set DND–R from the phone configuration window or the phone profile configuration window in Cisco Unified Communications Manager Administration.

When DND–R is enabled, the call is not presented to the terminal that has Call Reject enabled. There is no audible or visual indication of incoming calls on that end point. To enable DND–R, set the DND Status as true and the DND option to Call Reject.

FeaturePriority overrides DND. It can have any of the following values:

- 1: Normal

- 2: Urgent
- 3: Emergency

This release introduces FeaturePriority in connect() API on CiscoCall. FeaturePriority in selectRoute() and redirect() API is already supported in prior releases. When feature priority as EMERGENCY is specified in connect() API, and the destination terminal has DND–R enabled, the call still rings at the destination terminal and overrides the DND–R settings.

When a terminal has DND–R enabled and receives an intercom call, DND–R settings are overridden and call presents. This is because feature priority is always 2 (URGENT) for intercom calls.

In non- shared line scenario where A calls B and Terminal B has DND–R enabled, CallCtlConnFailedEv with cause USER_BUSY is delivered on A. Users would see the same behavior if DND–R is enabled on all the terminals that have shared DNs.

In the case of shared lines when at least one of the terminal does not have DND–R enabled and a call is placed to the shared line, Cisco Unified JTAPI delivers TermConnPassiveEv and CallCtlTermConnInUseEv for the terminals that have DND–R enabled (assuming the call was made with NORMAL feature priority). TermConnPassiveEv and CallCtlTermConnBridgedEv is delivered if DND–R is disabled on the terminal during a call.

A new event CiscoTermDNDOptionChangedEv will be sent to the terminal observer whenever the DND option changes on the phone window or Common Phone Profile window in Cisco Unified Communications Manager Administration.

Default DND option is Ringer–off and Route points do not support DND.

Interface Changes

CiscoTermDNDStatusChangedEv, on page 604; CiscoCall, on page 326; CiscoTermEvFilter, on page 608

Message Sequences

DND-R, on page 893

Backward Compatibility

This feature is backward compatible. Application will receive new events if this feature is configured. The new events are filtered through TerminalEventFilter interface (CiscoTermEvFilter). By default filter is disabled and the new events are not delivered.

Drop Any Party

This feature provides the capability to drop any participants from a conference call. Cisco Unified JTAPI allows applications to drop participants from conference using the existing interface Connection.disconnect() even if the application is not observing the address for the connection. Previously, applications could only disconnect connections for which Address is an observed Address.

Feature behavior varies based on the settings for the Cisco Unified Communications Manager service parameter Advanced Ad Hoc Conference Enabled. If this service parameter is set to False, applications can drop connections for an unobserved address in a conference call only if the application observes the conference controller's address. If this parameter is set to True, applications can drop connections without any restriction.

Cisco Unified JTAPI provides an interface on CiscoConnection to get an array of CiscoPartyInfo objects for the connection. CiscoPartyInfo is used to disconnect participants from a conference using a new interface, disconnect(), provided on CiscoConnection. A normal line has only one CiscoPartyInfo on its connection, but a shared line has one CiscoPartyInfo for each line in the shared line. This enables applications to selectively disconnect a shared line participant if more than one shared line participants are in the conference call. Since shared line participants have only one connection, if the application uses the existing Connection.disconnect() API, it drops all the shared line participants.

Cisco Unified JTAPI provides an interface setDropAnyPartyEnabled() on CiscoJtapiProperties to enable or disable this feature and by default, it is enabled. Alternatively, applications can have the JTAPI ini parameter dropAnyPartyEnabled = 0 in jtapi.ini file to disable Drop Any Party feature and dropAnyPartyEnable = 1 to enable this feature. If dropAnyPartyEnable parameter is not present in jtapi.ini file, the feature is enabled by default.

Cisco Unified JTAPI also provides an interface, isConferenceCall(), on CiscoCall to determine if a call is a conference call. This simple method returns a Boolean.

Interface Changes

See CiscoCall, on page 326 and CiscoConnection, on page 380

Message Sequences

See Drop Any Party Use Cases, on page 1178

Backward Compatibility

This feature is backward compatible.

Dynamic CTI Port Registration

This feature lets applications provide an IP address (ipAddress) and port number (portNumber) for each call or whenever media is established. To use this feature, applications must register the media terminal by supplying media capabilities. When a call is answered at this media terminal, CiscoMediaOpenLogicalChannelEv is sent to applications. This event gets sent whenever media is established. Applications must react to this event and specify the IP address and port number where media gets established.

A CiscoMediaTerminal represents a special kind of CiscoTerminal that allows applications to terminate RTP media streams. Unlike a CiscoTerminal, a CiscoMediaTerminal does not represent a physical telephony endpoint, which is observable and controllable in a third-party manner. Instead, a CiscoMediaTerminal represents a logical telephony endpoint, which may be associated with any application that terminates media. Such applications include voice messaging systems, interactive voice response (IVR), and softphones.

Note Only CTIPorts appear as CiscoMediaTerminals through Cisco Unified JTAPI.

Terminating media comprises a two-step process. To terminate media for a particular terminal, an application adds an observer that implements the CiscoTerminalObserver interface by using the Terminal.addObserver method. Finally, the application registers its IP address and port number to which the terminal incoming RTP streams get directed by using the CiscoMediaTerminal.register method.

To register the ipAddress and portNumber dynamically on a per-call basis, applications must register by only providing capabilities that they support. Applications must react to CiscoMediaOpenLogicalChannelEv that gets sent whenever media is established. If any features are performed before applications react to CiscoMediaOpenLogicalChannelEv, the features may fail.

If the applications do not respond to this event during the time that is specified in the Media Exchange Timer in the Cisco Unified Communications Manager Administration windows, the call may fail.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for Dynamic CTIPort Registration Per Call, see Message Sequence Charts, on page 755

Note The ChangeRTPDefaults interface is not supported on CiscoMediaTerminal.

The following new or changed interfaces exist for Dynamic CTIPort Registration Per Call:

Interface CiscoMediaOpenLogicalChannelEv Extends CiscoTermEv

int	**getpacketSize ()** Returns the packet size of the far end in milliseconds.
int	**getPayLoadType ()** Returns the payload format of the far end, one of the following constants:
CiscoRTPHandle	getCiscoRTPHandle () Returns the CiscoTerminalConnection object on which applications must invoke the setRTPParams request.

Interface CiscoRTPHandle

int	getHandle() Returns an integer representation of this object, currently the Cisco Unified Communications Manager CallLeg ID.

CiscoProvider

CiscoCall	getCall (CiscoRTPHandle rtpHandle) Returns the call object with the rtpHandle that is associated with a specific terminal. If no callobserver gets added to the terminal at the time when the applications receive CiscoRTPHandle in CallOpenLogicalChannelEv, CiscoCall may be null.

E911 Teleworker

The main purpose of this feature was intended to provide Location awareness for teleworkers and off premise users so that they can make emergency calls from off premises. The API's can also be used by all applications in a generic way as described below.

Primarily this feature adds two JTAPI API methods (selectRoute & redirect) that are overloaded to add an additional XML parmater to the list of their exisiting parameters. Any application can use these overloaded selectRoute() and redirect() methods to pass XML data to the call receiving side. The format of the XML data that can be passed is seen below:

```
<data>
     <item>
         <type>contact</type>
         <operation>append</operation>
         <protocol>SIP</protocol>
         <value>;+sip.instance = &quot;&lt;urn:uuid = *guid*&gt;&quot;</value>
     </item>
</data>
```

When an application sends XML data using one of the above API, CTI parses the data and extracts the text from the 'value' node in the XML and passes it on to CCM. CCM will then append this text to the outgoing SIP Invite message 'contact:' header. Once the end points like the SIP trunk or the SIP phone receives it, they can extract that data from the contact header and process it. Currently only SIP protocol's contact header field data is the only one supported but this can be expanded to include others headers fields and other protocols in future releases.

In the current release of CUCM only the following values for the XML nodes are supported: type: contact, operation: append, protocol: SIP. The value node in the above xml format is the one that carries the required application data to the end point.

The new parameter is a double byte array for overloaded selectRoute () Method to accommodate xml data for each selected routes and single byte array for the redirect () method. The parameter takes either a XML format String or a NULL value.

Interface Changes

CiscoRouteSession, on page 517, CiscoConnection, on page 380

Message Sequence

E911 Teleworker, on page 896

Backward Compatibility

This is a new feature and will be backward compatible

Enable or Disable Ringer

The CiscoAddress extension allows applications to set the status of the ringer for all lines on a device. No events generate when the ringer setting gets changed from the administration windows or anywhere else.

Encryption Enhancement

Unified Communications Manager Release 10.0(1) adds support for public key encryption which is more secure than the former symmetric key method. All JTAPI clients must be upgraded to the latest version bundled with Unified Communications Manager Release 10.0(1) to leverage this security enhancement. The JTAPI client is available under the "Applications-Plugins" menu from CCMAdministration.

The service parameter "Require Public key encryption" has been added. This parameter determines the encryption method required by Unified Communications Manager when authenticating CTI applications. When set to True, Unified Communications Manager requires CTI applications to authenticate using public key encryption; available in JTAPI client version 10 or later. When set to False (default), Unified Communications Manager allows CTI applications to authenticate by using either symmetric key or public key encryption. CTI applications must upgrade JTAPI/TSP client plugins to version 10.0(1) or later to authenticate when using public key encryption.

Although there are no interface changes for this enhancement, Cisco recommends that applications update CiscoJTAPI libraries to take advantage of this security enhancement.

Note No changes are required in the application layer. Applications need to update the Cisco JTAPI to the 10.x version to leverage the new security enhancement.

Note Cisco recommends that applications upgrade their Cisco JTAPI versions and set this service parameter to true. In future releases this service parameter will be deprecated.

Interface Changes

There are no interface changes for this feature.

Message Sequences

See Encryption Enhancement, on page 897.

Backward Compatibility

To maintain backward compatibility, a new CTI Manager service parameter is introduced:

"Require Public Key Encryption."

End to End Call Tracing

This feature enables the application to track any call uniquely. JTAPI associates a uniqueID with every Connection object. The same ID is exposed to the application through a new API getUniqueID(Terminal term) on the interface CiscoConnection. This uniqueID is only available for connection of observed addresses.

When a connection is created, the application can receive the uniqueID and write it in the Call Details Record. For Shared Line scenarios, each shared line has a uniqueID, which can be retrieved by passing the corresponding

Terminal to getUniqueID API. UniqueID may or may not be the same for different shared lines depending on the scenario. The application can query the uniqueID corresponding to each shared line on receiving TermConnActiveEv for that shared line Terminal.

Whenever the uniqueID changes, JTAPI delivers CiscoConnectionUniqueIDChangedEv to the call observer of the application.

Note As of Release 8.0(1), there is no supporting use case where JTAPI delivers CiscoConnectionUniqueIDChangedEv event to the application.

Interface Changes

CiscoConnection, on page 380, CiscoConnectionUniqueIDChangedEv, on page 393.

Message Sequences

End to End Call Tracing, on page 898

Backward Compatibility

This feature is backward compatible.

EnergyWise Deep Sleep Mode

This feature allows the phone to participate in an EnergyWise enabled system. The phone reports its power usage to a EnergyWise compliant switch to allow the tracking and control of power within the customer premise. The phone provides alternate reduced power modes including an extremely low, off mode. The Cisco Unified Communications Manager administrator configures and exclusively manages the phones power state through vendor specific configuration on the Cisco Unified CM Admin pages.

When the phone turns off power after negotiation with an EnergyWise switch, it unregisters from Cisco Unified CM and enters Deep Sleep/PowerSavePlus mode. Phones automatically re-register back with the Cisco Unified CM once the Deep Sleep mode configured PowerON time is reached.

However, you can press the ‘select’ key on the Cisco Unified IP Phones Series 9900 and 6900 while in Deep Sleep/PowerSavePlus mode to wake up the phone, these phones automatically power on and re-register back with the Cisco Unified CM. However, for Cisco Unified IP Phones 7900 Series phones, you can neither power on nor re-register back with the Cisco Unified CM during Deep Sleep/PowerSavePlus mode unless the ‘PowerON’ time is reached. You can configure Deep Sleep mode on the Device page of the Cisco Unified CM. Configure Deep Sleep mode for the phones at least 10 minutes before the actual power off time to allow the information to synchronize between the switch and the phone.

The Power off idle timer enables only in the case when there is physical interaction on the phone. For example if there is a call on the EnergyWise configured phone during the deep sleep time and the user tries to disconnect the call from the application, then the power off idle timer defaults to 10 minutes but if the user disconnects the call manually from the phone, then the power off idle timer takes the value configured on the Cisco Unified CM device page.

When a terminal unregisters from Cisco Unified CM, JTAPI exposes CiscoProvTerminalUnRegisteredEV event to application with a new reason “CiscoProvterminal UnRegisteredEV.ENERGYWISE_POWER_SAVE_PLUS”.

JTAPI sends CiscoTermOutOfServiceEv event to the application with the cause "CiscoOutOfServiceEv.CAUSE_ENERGYWISE_POWER_SAVE_PLUS" when a terminal goes out of service due to Deep Sleep mode configured.

JTAPI sends CiscoAddrOutOfServiceEv event to application with a new cause "CiscoOutOfServiceEv.CAUSE_ENERGYWISE_POWER_SAVE_PLUS" when an address goes out of service due to Deep Sleep mode configured.

Interface Changes

public interface CiscoProvTerminalUnRegisteredEv

When a terminal unregisters from the Cisco Unified CM because of Deep Sleep mode, JTAPI sends CiscoProvTerminalUnregisteredEv to the application with the reason "ENERGYWISE_POWER_SAVE_PLUS".

Field Summary	
public static final int	ENERGYWISE_POWER_SAVE_PLUS

Reason Codes

ENERGYWISE_POWER_SAVE_PLUS

```
Sample Code:public class MyTermObserver implements ProviderObserver {

  public void providerChangedEvent (ProvEv[] evlist) {
        for(int i = 0; evlist ! =  null && i < evlist.length; i++){
…
…
If ( evlisth[i] instanceof  CiscoProvTerminalUnregisteredEv){
CiscoProvTerminalUnregisteredEv ev  =  (CiscoProvTerminalUnregisteredEv)evlist[i];
if(ev.getReason()  =  =
CiscoProvTerminalUnregisteredEv.ENERGYWISE_POWER_SAVE_MODE){
System.out.println("Terminal Unregistered from CUCM because of deep
with the reason as ENERGYWISE_POWER_SAVE_PLUS
 ");
}
```

public interface CiscoOutOfServiceEv

When a terminal/address unregisters from the Cisco Unified CM because of deep sleep mode, Jtapi delivers CiscoTermOutOfServiceEv and CiscoAddrOutOfServiceEv to the application with this new cause "CAUSE_ENERGYWISE_POWER_SAVE_PLUS".

Field Summary	
public static final int	CAUSE_ENERGYWISE_POWER_SAVE_PLUS

Cause Code

CAUSE_ENERGYWISE_POWER_SAVE_PLUS

Message Sequences

Energywise Deep Sleep Mode, on page 917

Backwards Compatibility

This feature is backward compatible.

Extension Mobility Cross Cluster

This feature allows users to log in to an IP phone registered to a cluster with user profiles configured with another cluster. The Extension Mobility feature allows a user to log in to an IP phone to appear as user's desk phone temporarily, subject to the administrative policy. After logging on to an IP phone, the user can receive incoming calls normally routed to the user's desk phone and retain the personalized configuration, such as speed dials, services links and other user-specific properties.

Currently, Extension Mobility service is limited to a single Cisco Unified Communications Manager (Cisco Unified Communications Manager) cluster. A user provisioned in one cluster today cannot log in to an IP phone of another cluster with the Extension Mobility feature, even though both clusters may belong to the same enterprise. This limitation is overcome with the introduction of this new feature, which allows the user provisioned in one cluster to log in to an IP phone of another cluster.

With the existing behavior, when a user logs in to a terminal with a user ID that matches the user ID used by Cisco Unified Communications Manager JTAPI application, the terminal is treated as part of the control list and application is able to add call observer on the terminal and/or address.

As part this feature support, Extension Mobility profiles can be added to the user's control list via the Cisco Unified Communications Manager Admin pages. When a user uses Extension Mobily to log into a device using a profile in the control list, JTAPI delivers CiscoAddrCreatedEv and CiscoTermCreatedEv, and application can add call observer to control the terminal or address.

JTAPI exposes getCiscoCause () API on all provider events. For provider events associated with non-Extension Mobility login or logout scenarios, the cause delivered will be CiscoProvEv.NORMAL. For provider events associated with Extension Mobility login or logout scenario, the cause may be any of the below depending on the type of Extension Mobility login or logout:

```
CiscoProvEv.CAUSE_EM_LOGIN                    CiscoProvEv.CAUSE_EM_LOGOUT
 CiscoProvEv.CAUSE_EM_LOGIN_PROFILE_ADD CiscoProvEv.CAUSE_EM_LOGOUT_PROFILE_REMOVE
```

Interface changes explain more about each of these causes.

The following is a complete set of provider events that have API getCiscoCause():

```
CiscoAddrActivatedEvCiscoAddrActivatedOnTerminalEv
CiscoProvFeatureEv
CiscoProvTerminalCapabilityChangedEv
CiscoAddrRestrictedEv
CiscoTermActivatedEv
CiscoTermRestrictedEv
CiscoAddrCreatedEv
CiscoTermCreatedEv
CiscoAddrRemovedEv
CiscoTermRemovedEv
```

```
CiscoAddrAddedToTerminalEv
CiscoAddrRemovedFromTerminalEv.
```

JTAPI exposes getLoginType () on CiscoTerminal to indicate if the terminal is part of the Home or Visiting cluster when a user does an Extension Mobility login or logout. Accordingly, return value will be CiscoTerminal.NO_LOGIN, CiscoTerminal.NATIVE_LOGIN or CiscoTerminal.VISITOR_LOGIN.

Home Cluster is the Cisco Unified Communications Manager cluster from which the traveling EMCC user starts. This is the user's home cluster where the user profile resides.

Visiting Cluster is the Cisco Unified Communications Manager cluster which the traveling EMCC user visits. This is also the cluster that owns the phone at which the user does Extension Mobility login.

Interface Changes

CiscoProvEv, on page 475, CiscoTerminal, on page 611

Message Sequences

Extension Mobility Cross Cluster, on page 972

Backward Compatibility

This feature is backward compatible.

Extension Mobility Username Login

The Extension Mobility Login Username enables applications to get the Extension Mobility login username from the API provided on CiscoTerminal.

Interface Changes

CiscoTerminal, on page 611

Message Sequences

Extension Mobility Login Username, on page 1120

External Call Control

External Call Control enables Cisco Unified Call Manager (Cisco Unified Communications Manager) to route calls based on enterprise policies and presence-based routing rules of individual users. When call intercept is enabled, Cisco Unified Communications Manager queries the designated web services hosting the enterprise policies or user rules and routes the calls following the routing decisions returned.

Starting from Release 8.0(1), JTAPI supports wildcard routepoins, as well as translation patterns.

Interface Changes

CiscoCall, on page 326, CiscoConnection, on page 380, CiscoAddress, on page 283

Message Sequences

External Call Control, on page 923

Backward Compatibility

This feature is backward compatable. Existing applications will not be impacted by the changes for this feature. There are, however, implications and limitations to applications regarding Wildcard Routepoints as they exist today, which are being addressed by adding a service parameter, described below. Applications that do not use Wildcard Route Points will be completely unaffected by this development.

The first is that an application controlling wildcard routepoints used to get three JTAPI connections on a call. One with the caller, the second with the dialed Directory Number and the third with the wildcard routepoint that JTAPI is observing. The third connection has been removed with this feature.

The second backward compatability issue is with the various called party fields during a Wildcard Routepoint scenario. Before implementation of this feature, CiscoCall.getCalledAddress() and CiscoCall.getCurrentCalledAddress() both returned the actual dialed Directory Number, and it was not possibl eto retrieve the Wildcard Directory Number. After this fix, both CiscoCall.getCalledAddress() and CiscoCall.getCurrentCalledAddress() return the Wildcard Directory Number, while CiscoCall.getModifiedCalledAddress() returns the actual dialed Directory Number. This is a fix for an issue that built an errorenous call model in JTAPI, but it may cause applications using this feature in this way to break.

Both these issues have been addressed by adding a new service parameter at the CTI layer, known as Use WildCard pattern in CTI Call Info. This service parameter is set to OFF by default and continues the existing behavior. If an application wants to take advantage of the new information provided to it regarding Wildcard Routepoints, the service parameter must be changed to ON. This service parameter applies only when wild card Route Point is the called party. You must note that there are use cases for this feature that provide details of the Wildcard Routepoint scenario with the service parameter set to both ON and OFF, but the use case where it is set to OFF is currently not supported, shows the call flow as it exists today.

End to End Session ID for Calls

Cisco Unified Communications Manager generates a unique session identifier for each leg in a call. This feature enables the application to track a call end to end uniquely with a Session ID.

Cisco JTAPI exposes the following new methods for applications to get unique session identifiers for each connection in a call:

- CiscoConnection.getLocalUUID(TerminalConnection)
- CiscoConnection.getPeerUUID(TerminalConnection)

The methods accept a TerminalConnection object associated to that connection as a parameter, and return a String representing the UUIDs of the local and peer participant in a given CiscoConnection respectively. If a null object is passed as a parameter, the methods will return the UUID of the active TerminalConnection in the CiscoConnection.

The SessionID is acquired by merging the localUUID and the peerUUID in the following format:

```
device=<localUUID>;remote=<peerUUID>;
```

The Session ID is generated within Cisco Unified Communication Manager for non-SIP devices. SIP devices generate their own Session IDs and publish them in the SIP INVITE message to Cisco Unified Communication Manager. This information is visible to the application through the respective interface.

The Local Universal Unique Identifer (localUUID) for a CiscoConnection in a CiscoCall is generated in the peerUUID on the other side in the CiscoCall and vice versa.

This relation is assured for a basic two party call and is retained through the following features:

- Redirect
- Call Forward
- Transfer
- Hold/Resume

If the Cisco call is shared between multiple devices, the CiscoConnection.getPeerUUID(null) on the calling side will return the UUID of any of the available terminals while the CiscoConnection on the called side is in CiscoConnection.ALERTING state. Once the call is answered CiscoConnection.getPeerUUID(null) on the calling side will return the uuid of the active TerminalConnection.

Interface Changes

CiscoConnection, on page 380.

Message Sequences

End to End Session ID for Calls, on page 975

Backward Compatibility

This feature is backward compatible.

FIPS Compliance

This feature allows Unified Communications Manager to operate in Federal Information Processing Standard (FIPS) mode. FIPS specifies a minimum security level for cryptographic functions, limitations on how data is stored, and which algorithms are allowed to be used to encrypt sensitive information. These strictly defined requirements are important to government agencies, hospitals, and other customers who would be interested in a higher level of security.

To enable FIPS Compliance, Unified Communications Manager applications must request this mode when they download certificates with JTAPI and open a provider. When operating in FIPS, Unified Communications Manager experiences minimal performance loss, but this loss should only be witnessed during the certificate downloads and when you open a JTAPI provider. FIPS should not affect anything once the application is running.

Starting from release 8.6(1), JTAPI can be configured as FIPS Compliant.

Important Notes

In FIPS, there are two distinct “cryptographic entities”: The JTAPI application and the CUCM server machine (or cluster). The FIPS compliance of one does not, in any way, affect the other. Setting JTAPI to run in FIPS compliance encrypts the client-side certificates with a FIPS-compliant algorithm, and connect using only approved SSL/TLS algorithms. It will not make the CUCM server or cluster secure or FIPS compliant.

Likewise, having the CUCM operate in FIPS mode will not make JTAPI store certificates with FIPS-compliant algorithms. They are distinct items, separated by a"cryptographic boundary."

Also, even if JTAPI operates in FIPS-compliant mode, your application may not. Your applications must handle cryptographic information and other sensitive data with special attention in order to be FIPS-compliant.

As mentioned earlier, applications that need to use FIPS compliance must not only explicitly request it, but also download cryptographic libraries and modify their classpath variables to include them.

Until Unified Communications Manager release **12.5(1),** JTAPI used RSA libraries for FIPS-compliant operations. With release **12.5(1)** and later, JTAPI on Windows uses **RSA** libraries, while on Linux it uses **CiscoJ** libraries.

As of Unified Communications Manager release 14SU2, JTAPI uses **BCFIPS** libraries for all security-related operations. If configured to operate in FIPs mode, JTAPI moves **BCFIPS** libraries to approved only mode to enforce FIPS compliance.

The libraries are detailed below:

The RSA libraries are:

"jcmFIPS.jar" "cryptojcommon.jar", " cryptojce.jar" and "sslj.jar", are FIPS-compliant libraries from RSA, Inc.

The CiscoJ libraries are:

The CiscoJ libraries are "CiscoJCEProvider.jar", "log4j-1.2.17.jar", "slf4j-api-1.7.24.jar", "slf4j-log4j12-1.7.24.jar", "slf4j-simple-1.7.24.jar", "bcpkix-jdk15on-154.jar", and "bcprov-jdk15on-154.jar".

Note From release **12.5(1)SU5** on this train and up to **14SU1**, "bcpkix-jdk15on.jar" and "bcprov-jdk15on.jar" are used instead of "bcpkix-jdk15on-154.jar" and "bcprov-jdk15on-154.jar" respectively.

The BCFIPS libraries are:

"bc-fips.jar", "bcpkix-fips.jar", "bctls-fips.jar".

These libraries contain special implementations of several key cryptographic functions that replace the older implementation in jtapi.jar.

In case your application contains a lib folder where third-party libraries are stored, your classpath should look like the following.

- For the JTAPI plugin using RSA libraries (please refer above for library usage info as per the Unified Communications Manager release):

  ./libs/jcmFIPS.jar;./libs/cryptojcommon.jar;./libs/cryptojce.jar;./libs/sslj.jar;./libs/jtapi.jar

- For the JTAPI plugin using CiscoJ libraries (please refer above for library usage info as per the Unified Communications Manager release):

  ./libs/CiscoJCEProvider.jar;./libs/CiscoJUtils.jar;./libs/CiscoJCEJNI.so;./libs/libssl.so;

  ./libs/libssl.so.1.0.1;./libs/log4j-1.2.17.jar;./libs/libciscosafec.so;./libs/libciscosafec.so.4;

  ./libs/libciscosafec.so.4.0.1;./libs/libcrypto.so;./libs/libcrypto.so.1.0.1;

  ./libs/slf4j-api-1.7.24.jar;./libs/slf4j-log4j12-1.7.24.jar;./libs/slf4j-simple-1.7.24.jar;

  ./libs/bcpkix-jdk15on.jar;./libs/bcprov-jdk15on.jar;./libs/jtapi.jar

- For the JTAPI plugin using BCFIPS libraries (please refer above for library usage info as per the Unified Communications Manager release):

  ./libs/bc-fips.jar;./libs/bcpkix-fips.jar;./libs/bctls-fips.jar;./libs/jtapi.jar

Even with the classpath set this way, the JTAPI security code works the same way it does now unless the application specifically requests to run in FIPS mode.

To request that JTAPI run in a FIPS-compliant mode, applications must use some of the new methods introduced as part of this feature development and specify the new "fipsCompliant" parameter as **True**. For more information, see the following "Interface Changes" section.

Interface Changes

CiscoJtapiPeerImpl, on page 428, CiscoProvider, on page 486, and CiscoJtapiProperties, on page 429

Message Sequences

No impact.

Backward Compatibility

This feature is backward compatible. JTAPI, including secure providers, runs exactly as they do today, if the application does not specify that they wish to run in FIPS-compliant mode. This choice is deliberate; applications unaffected by FIPS compliance do not interact with FIPS compliance. No changes are required on the applications' part.

Applications that want to operate in a FIPS-compliant mode has to explicitly request it when downloading certificates with JTAPI, and when opening a provider. In addition, applications are required to download supplementary cryptographic libraries (jar files) from the CUCM server, and modify their classpath accordingly to include them before the jtapi.jar library.

Forced Authorization and Client Matter Codes

Forced Authorization Codes (FACs) force the user to enter a valid authorization code prior to extending calls to specified classes of dialed numbers (DN), such as external, toll, or international calls. Authorization information is written to the Cisco Unified Communications Manager CDR database.

Client Matter Codes (CMCs) let the user enter a code before extending a call. Customers can use Client Matter Codes for assigning accounting or billing codes to calls that are placed, and Client Matter Code information is written to the Cisco Unified Communications Manager CDR database.

Supported Interfaces

Cisco Unified JTAPI supports FAC and CMC in the following interfaces:

- Call.Connect()
- Call.Consult()
- Call.Transfer(String)
- Connection.redirect()

- RouteSession.selectRoute()

Call.Connect() and Call.Consult()

When an application initiates a call with one of these interfaces to a DN that requires an FAC, CMC, or both codes, CiscoToneChangedEv is delivered on a CallObserver that also contains which code or codes are required for the DN. The getCiscoCause() interface returns CiscoCallEV.CAUSE_FAC_CMC for this even if it is delivered because of FAC_CMC feature. The getTone() interface returns CiscoTone.ZIPZIP to indicate that a ZIPZIP tone played.

Upon receiving the CiscoToneChangedEv, applications need to enter the appropriate code or codes by using the connection.addToAddress interface with a # terminating string. Digits either can be entered one at a time within the interdigit timer value (T302 timer) for each digit including the # terminating character, or all the digits, including the # termination character, can be entered within the T302 timer value that is configured in Cisco Unified Communications Manager Administration.

When FAC and CMC Are Both Required

For a DN that requires both codes, the first event is always applies for the FAC, and the second code applies for the CMC, but the application can send both codes, separated by a pound sign (#), in the same request. The second event remains optional, based on what the application sends in the first request.

The application can send both codes at the same time, but both codes must end with #. as shown in the following example:

```
connection.addToAddress("1234#678#")
```

where 1234 represents the FAC and 678 specifies the CMC.

In this case, the application does not receive a second CiscoToneChanged.

The first CiscoToneChangesEv will have getWhichCodeRequired() = CiscoToneChanged.FAC_CMC_REQUIRED, and getCause() = CiscoCallEv.CAUSE_FAC_CMC.

In response, one of the following cases can occur:

- The application sends FAC and CMC in the same connection.addToAddres(code1#code2#) request. In this case, no second CiscoToneChangedEv gets sent to the application.
- The application sends only a FAC code in connection.addToAddress(code#1). In this case, the application receives a second CiscoToneChangedEv with getWhichCodeRequired() = CiscoToneChangedEv.CMC_REQUIRED.
- The application sends only part of the first code or the complete first code and incomplete second code (if the code is not terminated with #, it remains is incomplete and the system waits for the T302 timer to expire and tries to validate the code). If the code is incomplete, a second CiscoToneChangedEv tone gets generated with getWhichCodeRequired() = CiscoToneChangedEv.CMC_REQUIRED and getCause() = CiscoCallEv.CAUSE_FAC_CMC.

PostCondition Timer

The PostCondition timer resets each time that the connection.addToAddress interface is invoked to send code. FAC and CMC must have the terminal # [for example, Connection.assToAddress("1234#"), where 1234 is the FAC]. The system waits for the T302 timer to expire, then extends the call if all codes have been entered.

If all codes have not been entered, the system plays reorder tone. In this case, the application could receive PlatformException with postConditionTimeout even if the call is extended. To avoid this, the application needs to increase the postcondition timeout by using JTAPI Preferences.

If the application uses call.connect() or call.consult() to initiate a call, but the FAC or CMC (including #) is not entered from a Cisco Unified IP Phone within the postcondition timeout limit, the request could get a platformException with postCondition timeout, but the call may actually get extended. To avoid this, the application needs to increase the postcondition timeout by using JTAPI preferences.

Shared Lines

If the initiating party is a shared line, applications need to use setRequestController to set active terminalConnection before passing additional digits by using the connection.addToAddress interface.

Invalid or Missing Codes

If a code is invalid or no code is entered before the T302 timer expires, the call gets rejected with callCtlCause cause code as CiscoCallEv.CAUSE_FAC-CMC.

Call.transfer(String) and Connection.redirect()

Two additional string parameters (facCode, cmcCode) are added to these interfaces to support FAC and CMC. The default value for these codes represent null values.

No CiscoToneChangedEv gets delivered for these requests for DNs that require codes. A call that is conditionally redirected to a DN, a FAC, a CMC, or both, does not get rejected but remains connected if either code is incorrect.

RouteSession.selectRoute()

Two additional arrays of string parameters (facCode, cmcCode) support FAC and CMC. For each routeselect element, applications can specify the code for the DN. Applications need to specify null values for DNs that do not require any codes. The default values for the codes are null values.

If one routeselected element does not contain the correct code, the next element in the arrays gets tried. If all of them fail, reRouteEvent gets sent to the application.

Note The system does not support forwarding to a DN that requires an FAC or CMC code. The application can set the forward number to these DNs by using the Address API, but calls forwarded to these numbers are rejected.

Forwarding on No Bandwidth and Unregistered DN

This feature enhances the forwarding logic to handle the No Bandwidth & Unregistered DN cases:

- No Bandwidth: When a call cannot be delivered to a remote destination due to no bandwidth, the system reroutes the call to the AAR Destination Mask or voice mail. The user makes these configuration changes from the directory number window of the Cisco Unified Communications Manager GUI.

- Unregistered DN: When a call is placed to an unregistered DN, the system delivers the call to a DN that is configured for Call Forward on No Answer (CFNA).

When a call is forwarded due to Call Forward No Bandwidth (CFNB) to another cluster destination over a trunk/gateway that is using QSIG, call history might get lost. For example, if Phone A calls Phone B, which is in a low bandwidth location, with CFNB set to forward calls to Phone C, which is in a different cluster, and the QSIG protocol is used for this intercluster forwarding, then the original called party and the last redirecting party might not get passed to the destination party.

GetCallID in RTP Events

GetCallID provides an interface on RTP events to access any call information, such as calling party or called party, so applications can link RTP events with the calls.

The callLegID that is received in the RTP events from CTIManager gets used to determine the ICCNCall on the client side. This call passes on to the JTAPI layer, and the CiscoCall gets determined, from which CiscoCallID is obtained. This information gets used to construct the RTP events that are delivered to the application.

The following interface gets added to **`CiscoRTPInputStoppedEv`**, **`CiscoRTPInputStartedEv`**, **`CiscoRTPOutputStoppedEv`**, and **`CiscoRTPOutputStartedEv`**:

```
{ public CiscoCallID getCallID();
```

}

GetCallInfo

GetCallInfo interface on address provides applications with the ability to query CallInfo on an address. A query returns the CiscoAddressCallInfo object, which contains information about the number of active or held calls, maximum number of active or held calls, and the call object for current calls on the address. This interface also specifies what calls are at a specific address at a specific time.

Use the following interface to get information about calls that are present at the terminal:

```
{ public CiscoAddressCallInfo getAddressCallInfo(Terminal iterminal);}
```

GetGlobalCallID

GetGlobalCallID provides an interface on the CiscoCallID to get the nodeID and the Global Call ID (GCID) of the call; this exposes the GCID information that is available in the internal call object.

The following methods get added to the CiscoCallID interface:

```
{    /**
     * returns the callmanager nodeID of the call
     */
public int getCallManagerID();
```

```
     /**
     * returns the GlobalCallID of the call
     */
public int getGlobalCallID ();
}
```

}

Hairpin Support

A hairpin call happens when the call leaves one cluster to some other device across the gateway, then comes back to a device in the same cluster. The GCID for the call coming back into the cluster would differ from the GCID that originally initiated the call, even though both are in the same cluster. In previous releases, if JTAPI controlled both parties, there were two connections: one for CiscoAddress.Internal and the other for CiscoAddress.External.

JTAPI supports hairpin calls when an application monitors both ends of the hairpin call. Previously, only one end of the hairpin call could be monitored because the address was represented only as a DN.

In the current release, if two addresses exist with the same DN but one is within the same cluster and the other is across the gateway, JTAPI creates a separate address object for the external DN, and only one connection is returned for an address, based on its type. This process avoids hairpin issues, as in previous releases when the address was represented only as a DN and when an application retrieved connections for the address it used to get two connections.

Since fixing these issues could have caused compatibility issues with previous releases, a generic solution for these issues was developed in this release. Calls that involve an external party with the same DN as the monitored local party are now properly supported; however, no new interface is added for this feature.

Backward Compatibility

This feature is not backward compatible.

Half-Duplex Media Support

Currently JTAPI media events CiscoRTPInputStarted, CiscoRTPOutputStarted, CiscoRTPInputStopped and CiscoRTPOutputStopped do not indicate whether media is half duplex (receive only / transmit only) or full duplex (both receive and transmit).

This enhancement adds the capability to provide this information in a JTAPI media event. JTAPI provides an interface on the above media events to query whether media is half duplex or full duplex.

The half duplex media support feature does not impact JTAPI backward compatibility.

A new interface getMediaConnectionMode() is added to Cisco Unified JTAPI RTP events. This interface will return the following values depending on the media:

- CiscoMediaConnectionMode.NONE
- CiscoMediaConnectionMode.RECEIVE_ONLY
- CiscoMediaConnectionMode.TRANSMIT_ONLY

- CiscoMediConnectionMode.TRANSMIT_AND_RECEIVE.

CiscoRTPInputStarted/StoppedEv should only return RECEIVE_ONLY and TRANSMIT_AND_RECEIVE. The interface should not return NONE or TRANSMIT_ONLY. Ifthat happens, applications should ignore the event or log an error.

CiscoRTPOutputStarted/StoppedEv should only return TRANSMIT_ONLY and TRANSMIT_AND_RECEIVE. The interface should not return values NONE or RECEIVE_ONLY. Ifthat happens, applications should ignore the event or log an error.

CiscoMediaOpenLogicalChannedEv should only return RECEIVE_ONLY and TRANSMIT_AND_RECEIVE. The interface should not return values NONE or TRANSMIT_ONLY. Ifthat happens, applications should ignore the event or log an error.

public interface **CiscoRTPInputStartedEv**

int	`getMediaConnectionMode()` Returns CiscoMediaConnectionMode

public interface **CiscoRTPOutputStartedEv**

int	`getMediaConnectionMode()` Returns CiscoMediaConnectionMode

public interface **CiscoRTPInputStoppedEv**

int	`getMediaConnectionMode()` Returns CiscoMediaConnectionMode

public interface **CiscoRTPOutputStoppedEv**

int	`getMediaConnectionMode()` Returns CiscoMediaConnectionMode

Hold Reversion

The Hold Reversion feature provides applications with a notification when Cisco Unified Communications Manager notifies an address about the presence of a held call, when the call has been ONHOLD for a certain amount of time. Applications receive this notification as the CiscoCallCtlTermConnHeldReversionEv call control terminal connection event on their call observers on the particular address that has put the call ONHOLD. This notification is provided only once for the applications for the held call.

The event is sent only to the terminal connection of the terminal where the call was put on hold. If the address represents a shared line address, other terminal connections of the shared line address will not receive the event.

To receive this event, applications must add a call observer to the address. The cause for this event will be CAUSE_NORMAL. If the call observer is added after the hold reversion timer has expired and the notification

has already been sent to the phone, applications will receive CiscoCallCtlTermConnHeldReversionEv with cause CAUSE_SNAPSHOT.

For more information, see CiscoCallCtlTermConnHeldReversionEv, on page 346.

Hunt List

This feature enables the JTAPI application to observe addresses and terminals that are HuntList LineGroup members. Calls can arrive at these addresses either by another address calling it directly or through HuntPilot. When a call is made to HuntPilot, JTAPI creates a CiscoHuntConnection to represent HuntPilot and provides a Call Model that gives applications the information that the call is routed through HuntPilot. When a call is routed through HuntPilot and is connected to LineGroup Member, JTAPI call has three connections, two regular connections for calling and called addresses, and one CiscoConnection to HuntPilot through which that call was routed.

HuntPilot is not an observable address. The address representing Hunt Pilot is created when a call is made to a Hunt Pilot and is removed when the call is over. Applications cannot receive the Hunt Pilot address from the provider by using the getAddress() method.

In normal Hunt List calls, there are three connections, Calling, Hunt Pilot, and Hunt Member. When a call is made to the Hunt Pilot Directory Number, the call is offered to one of its members depending on the algorithm. The initial state of the call is Offering at the member. If members are not observed, the connection to Hunt Pilot goes through the normal states as CallCtrlConnection or Connection. If members are observed, connection to member goes to Offering state and the connection to Hunt Pilot goes to Established state. Applications must use the states of the observed party to track the state of the call.

call.getCurrentCalledParty() for a call to Hunt Pilot returns an address of type CiscoAddress.HUNT_PILOT. If the Hunt List member is the called address and is not observed by the application, connection to the member is created only when the call is answered.

CiscoHuntConnection extends CallControlConnection and can get into states that a call control connection could transition to expect for the network states.

Hunt pilots are represented by CiscoAddress objects and getType () would return CiscoAddress.HUNT_PILOT.

Only addresses returned for Cisocall.getCurrentCalledAddres () and CiscoCall.getCurrentCallingAddress () will have CiscoAddress.HUNT_PILOT type.

When calls to hunt pilot are involved in transfer or conference operations CiscoTransferStartEv, CiscoTransferEndEv, CiscoConferenceStartEv and CicsoConferenceEndEv are not delivered. Applications should use CiscoCallChangedEv to identify surviving call.

If consult calls or final call have CiscoHuntConnection, the application should not expect Transfer or Conference start and end events.

When configured in broadcast mode, all Hunt List members ring simulatenously. In JTAPI, call connections and terminal connections for Hunt List members are created only for members observed by the application.

Applications must enable this feature using the setHuntListFeatureEnabled (boolean) on CiscoJtapiProperties. This feature is disabled by default and applications are encouraged to adapt to the above call model and enable the feature using setHuntListFeatureEnabled () API. Observing a hunt list member without enabling the feature using setHuntListFeatureEnabled () is not a supported configuration and if observed, results in inconsistent call model and events. setHuntListFeatureEnabled() is introduced to enable applications that are currently using unsupported call scenarios with Hunt List to migrate to a supported model without breaking the existing functionality.

Applications can also enable this feature by adding, HuntListEnabled = 1, to jtapi.ini file and restarting the application.

Interface Changes

CiscoHuntConnection, on page 405, CiscoConnection, on page 380, CiscoAddress, on page 283, CiscoJtapiProperties, on page 429

Message Sequences

Hunt List, on page 994

Backward Compatibility

This feature is backward compatible.

Hunt List Connected Number

In Cisco Unified CM 9.0, the support for hunt pilots is enhanced to expose the connected number as the modifiedCalledAddress in a call involving a hunt pilot.

With this enhancement, when a user calls a hunt pilot and the call is answered by the hunt member L1, call.getModifiedAddress() returns the address of the member L1, whereas call.getCurrentCalledAddress() returns the address of hunt pilot. Before the call is answered, both these values will return the address of hunt pilot.

Interface Changes

There are no interface changes for this feature.

Message Sequences

See Hunt List Connected Number, on page 1037

Backward Compatibility

This feature is backward compatible. To enable this feature, a new Hunt Pilot configuration, "Display Line Group Member DN as Connected Party" is introduced. Application may choose to enable or disable feature based on their requirements. By default, this feature is disabled.

Hunt Log Status

With this feature, the Cisco JTAPI interface includes the ability of a terminal to sign in and sign out of the hunt group through CTI applications. Previously, this functionality was only available from Cisco Unified CM Administration interface.

Once a terminal is logged into a hunt group, it is able to receive calls which are offered on the line group where the line of terminal is associated.

Cisco Terminal is enhanced with two new methods:

- CiscoTerminal.getHuntLogStatus()

- CiscoTerminal.setHuntLogStatus()

These two new methods are used to get and set the value of huntLogStatus and the three new constants CiscoTerminal.DEVICE_HUNT_LOGGED_IN, CiscoTerminal. DEVICE_HUNT_LOGGED_OUT and CiscoTerminal. DEVICE_HUNT_NOT_APPLICABLE. The value is CiscoTerminal.DEVICE_HUNT_LOGGED_IN by default for any terminal that has the ability to log in to the hunt group.

A new interface, CiscoTermHuntLogStatusChangedEv, is introduced for applications to be notified with the event CiscoTermHuntLogStatusChangedEv when the value of hunt log status is changed and the filter is set.

CiscoTermEvFilter is enhanced with two new methods: CiscoTermEvFilter. setHuntLogStatusChangedEvFilter(boolean filterValue) and CiscoTermEvFilter.getHuntLogStatusChangedEvFilter() to set and get the value of filter, if the application wants to be notified by the event CiscoTermHuntLogStatusChangedEv the filter should be set to true. The value of filter is false by default.

Note The above methods are invoked only on devices which have observers added on it and the terminal object is in service.

Interface Changes

CiscoTermHuntLogStatusChangedEv, on page 666

CiscoTerminal, on page 611

CiscoTermEvFilter, on page 608

Message Sequences

Hunt Log Status for Phone Devices, on page 914

Backward Compatibility

This feature is backward compatible.

Intercom

The Intercom feature allows one user to call another user and have the call answered automatically with one-way media from the caller to the called party, regardless of whether the called party is busy or idle. The called user can press the talk back softkey (unmarked key) on their phone display, or the called user can invoke the join() JTAPI API, that is provided on TerminalConnection, to start talking to the caller. Only a specially configured intercom address on the phone can initiate an intercom call. Cisco Unified JTAPI creates a new type of address object named CiscoIntercomAddress for intercom addresses that are configured on the phone. The application can get all the CiscoIntercomAddresses that are present in the provider domain by calling the interface getIntercomAddresses () on CiscoProvider.

An intercom call can be initiated from the Cisco Unified JTAPI interface by calling the CiscoIntercomAddress.ConnectIntercom () interface. The application provides an intercom target DN for this interface. If the intercom target DN is preconfigured or preset by the application, the application can get the

target DN by calling the CiscoIntercomAddress.getTargetDN() interface; otherwise, the application must provide a valid intercom target for the call to be successful.

An intercom call is autoanswered at the intercom target; Cisco Unified JTAPI will move TerminalConnection/CallCtlTerminalConnection at the intercom target to the Passive/Bridged state. The application can invoke a join () interface on the TerminalConnection of the intercom target to initiate talk back. If join () is successful, the TerminalConnection/CallCtlTerminalConnection of the intercom target will move to an Active/Talking state. For an intercom call, Cisco Unified JTAPI only supports the following interfaces:

- Call.drop ()
- Connection.disconnect ()
- CallCtlTerminalConnection.join ()

The application cannot perform any feature operations on an intercom call. Cisco Unified JTAPI will throw an exception if the application invokes redirect, consult, transfer, conference, or park for a Connection on a CiscoIntercomAddress. The application will also receive an exception if it tries to invoke setForwarding (), getForwarding (), cancelForwarding (), unPark (), setRingerStatus (), setMessageWaiting (), getMessageWaiting (), setAutoAcceptStatus (), or getAutoAcceptStatus () on CiscoIntercomAddress.

Applications can get the value of a configured intercom target DN and the label on a CiscoIntercomAddress from the provided API. Cisco Unified JTAPI provides two types of APIs: one to return the default and another to return the current value set for the intercom target. The default value is the intercom target DN and label that are preconfigured through Cisco Unified Communications Manager Administration. The current value is the interim target DN and label that the application sets. If the application has not set any value, the current value remains the same as the default value. Applications can invoke the API setIntercomTarget () on CiscoIntercomAddress to set the intercom target DN, label, and unicode label. Only one application can set the intercom target, label, and unicode label for an intercom address. If two applications try to set the value, the first succeeds, and the second receives an exception. When a intercom target DN and label changes, Cisco Unified JTAPI provides a CiscoAddressIntercomInfoChangedEv to the AddressObserver that is added to CiscoIntercomAddress. If the application has set an intercom target DN and label, and a JTAPI or CTI failover or failback occurs, JTAPI or CTI will restore the previously set value of the intercom target DN, label, and unicode label. If the JTAPI or CTI cannot restore the intercom target DN, label, or unicode label, Cisco Unified JTAPI provides a CiscoAddrIntercomInfoRestorationFailedEv to the AddressObserver on CiscoIntercomAddress. In the case of an application failure, or if for any reason the application goes down, the target DN, label, and unicode label will reset to the default. JTAPI provides the interface resetIntercomTarget () on the CiscoIntercomAddress to reset the intercom target.

Auto-answer always stays enabled for CiscoIntercomAddress. The application can invoke the method getAutoAnswerEnabled () on CiscoAddress to get the auto-answer capability of an address.

For an intercom target that is connected with one-way media to the Intercom initiator, the device state would be set to CiscoTermDeviceStateWhisper. This is a new device state for the terminal object. In this state, the terminal can initiate a new call or receive a new incoming call. If the application enables a filter to receive this device state, the application receives CiscoTermDeviceStateWhisperEv. The application can enable a filter by calling setDeviceStateWhisperEvFilter() on CiscoTermEvFilter. The DeviceStates DEVICESTATE_ACTIVE, DEVICESTATE_HELD, and DEVICESTATE_ALERTING all override DEVICESTATE_WHISPER; if one call exists in active, held, or alerting state, and another in whisper, the DeviceState will be DEVICESTATE_ACTIVE, DEVICESTATE_HELD, or DEVICESTATE_ALERTING, respectively.

Note The Cisco Unified JTAPI implements the javax.telephony.TerminalConnection interface join() to let the intercom target talk back to the initiator. The system implements this interface for CiscoIntercomAddresses only. If applications invoke this interface for regular shared lines in a passive or bridged state, JTAPI throws a MethodNotImplimented exception.

Tip This feature is backward compatible if the application-controlled devices (terminals) do not have intercom lines configured on them. Applications can disable the intercom feature by not having an intercom line configured on the application-controlled devices (terminals).

For detailed information about these interface changes, see the following topics:

- CiscoHuntConnection, on page 405
- CiscoAddrIntercomInfoRestorationFailedEv, on page 305
- Related Documentation, on page 283
- CiscoCall, on page 326
- CiscoProvider, on page 486
- CiscoTermEvFilter, on page 608
- CiscoTerminal, on page 611
- CiscoTerminalConnection, on page 630
- CiscoTermDeviceStateWhisperEv, on page 601

Intercom Support for Extension Mobility

In Release 6.0(1) of Cisco Unified Communication Manager, support for intercom feature was added. Intercom feature requires destination to be auto-answered with one-way audio; therefore, no shared addresses can be configured for intercom. When user logs in by using Extension Mobility (EM) profile, it is possible to end up with shared address for intercom; so, currently extension mobility is not supported with intercom. Due to the wide use of extension mobility, this CIA is addressing the need to support intercom for extension mobility while still maintaining the single destination nonsharable nature of intercom addresses.

This feature requires intercom addresses to be configured with default terminal, and allows configuring of intercom address on EM profile. When EM user logs in to a terminal with EM profile that is configured with an intercom address, intercom address is available only if default terminal of intercom address is same as terminal where user has logged in. If an intercom address is configured on terminal but default terminal for intercom address is not that terminal, intercom address does not appear on terminal. If this terminal is configured in the control list of Cisco Unified JTAPI application, JTAPI does not create intercom address in the provider domain. From Cisco Unified JTAPI point of view, there is no new interface or changes to support this feature. However, this feature introduces some transitional scenarios where intercom functionality may not work on intercom addresses. See the use cases.

Backward Compatibility

This feature is backward compatible.

IPv6 Support

This feature provides support for IPv6 addresses and CiscoUnified JTAPI is enhanced to support IPv6 connectivity to CTIManager. It enables Cisco Unified JTAPI applications to see the IPv6 address as the calling party address if the IPv6 support feature is enabled and if the Calling Party is using an IPv6-enabled phone. This feature support the following functions:

- Cisco Unified JTAPI exposes new API canSupportIPv6() on the CiscoProviderCapabilities Interface to indicate whether Cisco Unified Communications Manager configuration is supporting IPv6.
- Cisco Unified JTAPI closes the media or route terminal if there is mismatch between what has been previously registered and what is currently configured. CiscoTermRegistrationFailedEv and the new reason code IP_ADDRESSING_MODE_MISMATCH are then sent as per this scenario.
- The IP Address capability of the Terminal is exposed by API getIPAddressingMode() on the CiscoTerminal Interface. The IP Address capability is available on CiscoTerminal/CiscoMediaTerminal and CiscoRouteTerminal.
- The IPv6 calling party IP address is provided through the Cisco extensions of CallCtlConnOfferedEv and RouteEvent in an InetAddress object as well as the IPv4 address for IPv4-enabled devices.

The RTP Address in CiscoRTPOutputStartedEv and CiscoRTPInputStartedEv also has an IPv6 address in case the observed device is an IPv6 device. That is, the API getLocalAddress() on CiscoRTPInputProperties and the API getRemoteAddress() on CiscoRTPOutputProperties can now return an IPv6 format IP Address. The API returns an InetAddress object, and applications can verify that it is an instance of Inet4Address or Inet6Address to determine if it is an IPv4 or IPv6 format IP Address.

Applications must reset the devices after their IP Addressing Mode is changed, otherwise there might be ambiguity in the expected results.

From Release 7.1, Cisco Unified JTAPI provides getIPAddressingMode() API on CiscoTerminal. The getIPAddressingMode() API for CTI Ports and Route Points are also supported from this release.

Cisco Unified JTAPI extends the same API on CiscoTerminal and it returns the configured IP addressing mode of the IP phone on the Cisco Unified Communications Manager Admin pages. If the user modifies the IP Addressing mode from the Cisco Unified Communications Manager Admin pages after the device is registered, the device must be reset. The updated value from Cisco Unified JTAPI is exposed only after the IP phone is reset. If the configured IP Addressing mode supports both IPv4 and IPv6 addresses, the phone may be registered with either of these addresses or with both. This depends on conditions such as network type and Cisco Unified Communications Manager support for IPv6. So, if the IP Addressing mode mode supports both IPv4 and IPv6 addresses, getIPAddressingMode() on CiscoTerminal returns CiscoTerminal.IP_ADDRESSING_MODE_IPV4_V6.

Interface Changes

See CiscoTerminal, on page 611

Message Sequences

See IPv6 Support, on page 1123 and IPv6 Support, on page 1230

Backward Compatibility

This feature is backward compatible.

iSac Codec

This enhancement provides support for iSac codec and enables the application to register CiscoMediaTerminal or CiscoRouteTerminal with iSac codec capability. For this codec, frame size and bit rate are variable and determined dynamically. Applications do not set these values.

The bit rate and packetSize that are exposed on interface CiscoRTPInputProperties and CiscoRTPOutputProperties will not be a constant for this codec, so application logic should not rely on these values if codec (payloadType) is iSac.

Interface Changes

See CiscoIsacMediaCapability, on page 409

Message Sequences

iSac Codec, on page 1051

Backward Compatibility

This feature is backward compatible.

Java Socket Connect Timeout

The Java Socket Connect Timeout enhancement enables the configuration of a timeout in seconds by using the Cisco Unified JTAPI specification and prevents connection delays to the CTIManager when the primary CTI Manager. The default is 15 seconds.

If the default of 15 seconds is unacceptable to the application, the default JAVA API of zero (0) sets the behavior to the normal JAVA Socket Connect API.

The values range from 5 through 180 seconds. Zero defaults to Java behavior of the socket connect without any time-out for connection.

Interface Changes

See CiscoJtapiProperties, on page 429.

Message Sequences

See CiscoJtapiProperties, on page 1122.

Backward Compatibility

This feature is backward compatible.

Join Across Lines

In this version, this feature allows applications to conference two calls that are on different addresses of the same terminal. It will also let applications add participants to a conference using a noncontroller. Join across lines is not supported on CTI-supported phones that run SIP.

You can disable join across lines feature by turning off Join Across Lines Policy service parameter, while you can disable Conference Chaining and feature to allow noncontroller adding participant to conference by disabling the "Advanced Ad Hoc Conference Enabled" and "Non-linear Ad Hoc Conference Linking Enabled" service parameters.

Note Join Across Lines is supported only on phones that run SCCP.

Interface Changes

There are no interface changes for this feature. Applications can use the current conference interfaces to conference calls on different addresses on the same terminal.

Backward Compatibility

This feature is backward compatible.

Join Across Lines (Only SCCP)

The Join Across Lines feature allows support for conference across lines. It allows two or more calls on different addresses of the same terminal to be joined though the join softkey on the phone or conference() API that JTAPI provides. The behavior to JTAPI applications change, as applications do not perceive a common controller in final and consult calls.

There is no change in the API and the same events are delivered whether calls are conferenced on the same address (regular conference) or across addresses (Join across lines). When join across lines feature is performed CiscoConferenceStartEv/EndEv will be provided to all addresses on the controller terminal that have consult or final calls that are being joined together into one conference.

In CiscoConferenceStartEv, the conferenceControllerAddress will always be the primary controller address. Application can now set the controller via the setConferenceController() API. If application does not specify this, then JTAPI itself would find a suitable controller for the conference. Cisco recommends that applications set the controller address when Join Across Lines feature is invoked.

If observer is not added on the controller address, applications may see null values for either the talking or held terminal connection values in the CiscoConferenceStartEv. Before this release, when application tried a conference across lines, the request failed at the JTAPI layer itself. With this release, the conference() API implementation enhances all requests to pass through after finding suitable terminal connections of the final and consult calls. JTAPI relies on the common terminal of the addresses involved in the call to find suitable terminal connections. Multiple conference across address is also supported when more than two calls need to be joined. SIP devices in 5.1.2 release do not support this feature. JTAPI throws exception (ILLEGAL_HANDLE) if this feature is requested on a SIP device.

There are no interface changes for this feature. Behavior changes with respect to events provided to applications.

Backward Compatibility

This feature is backward compatible, as there are no changes in the behavior of conference when this feature is not enabled. You can enable or disable this feature on a per-device basis. If the Join Across Lines setting on the device is set to Default, the system-wide CallManager service parameter Join Across Lines Policy setting is used. If this feature is enabled and application does a join across lines, there is a difference in behavior as stated.

JTAPI applications written for Release 5.1 should be backward compatible with JTAPI that was released with Release 5.1.2. Consider a JTAPI client upgrade only if new features are used.

Join Across Lines or Connected Conference Across Lines

User experience is enhanced in this release by introducing Cisco Unified IP Phone models that fall outside the purview of existing Join Across Lines service parameter. For these phones this feature is always enabled, without any service parameter to turn it off. For a detailed feature description, information about interface changes, and use cases, see Join Across Lines with Conference Enhancements (SCCP and SIP), on page 113.

Usage Guidelines

The points below indicate how applications must use the Direct Transfer Across Lines feature:

- Applications must add Call Observer on the both the lines across which they try join across lines or connected conference.
- Earlier, applications were recommended to check if both the calls have a common address and if that common address is on the same Terminal. For Join Across Lines, it is not required to check if the address is comoomon between two calls across which direct conference is invoked. It must be ensured that both the calls should each have an address that exists in common terminal.
- Cisco Unified JTAPI reports the same set of events, as it does currently, for conferencing of calls on the same address. Applications are not required do anything with these calls after invoking Conference() until receiving CiscoConferenceEndEv.
- As conference is done across addresses, applications do not get a common controller in CiscoConferenceStartEv and should upgrade the application logic. See Event Flow Comparison and Sample Code, on page 109 for details.

Event Flow Comparison and Sample Code

The following table provides details of the event flow also sample code.

Table 3: Event Flow Comparison and Sample Code for Conference Invocation

Join on Same Lines	Join Across Lines
Setup	
Address A on Terminal T1 Address B1, B2 on Terminal T2 Address C on Terminal T3	Address A on Terminal T1 Address B1, B2 on Terminal T2 Address C on Terminal T3
Feature Invokation	

Join on Same Lines	Join Across Lines
A calls B1[GC1 = GolbalCallID1]	A calls B1[GC1 = GolbalCallID1]
GC1: Connection A1 Conn1	GC1: Connection A1 Conn1
GC1: Connection B1 Conn2	GC1: Connection B1 Conn2
B1 calls C[GC2 = GolbalCallID2]	B2 calls C[G = GolbalCallID2]
GG2: Connection B1 Conn3	GG2: Connection B2 Conn3
GC2: Connection C Conn4	GC2: Connection C Conn4
GC1.conference(GC2)	GC1.conference(GC2)
Events Delivered to Application (assuming all parties are observed)	
GC1:	GC1:
CiscoConferenceStartEv	CiscoConferenceStartEv
[getConferenceControllerAddress() returns B1]	[getConferenceControllerAddress() returns B1]
ConnCreatedEv for C	ConnCreatedEv for C
ConnConnectedEv for C	ConnConnectedEv for C
CallCtlConnEstablishedEv for C	CallCtlConnEstablishedEv for C
TermConnCreatedEv for T3(Address C)	TermConnCreatedEv for T3(Address C)
CiscoConferenceEndEv	CiscoConferenceEndEv
GC2:	GC2:
CiscoConferenceStartEv	CiscoConferenceStartEv
[getConferenceControllerAddress() returns B1]	[getConferenceControllerAddress() returns B1]
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B2)
CallCtlTermConnDroppedEv for T2(Addresss B1)	CallCtlTermConnDroppedEv for T2(Addresss B2)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B2
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B2
TermConnDroppedEv for T3(Address C)	TermConnDroppedEv for T3(Address C)
ConnDisconnectedEv for C	ConnDisconnectedEv for C
CallCtlConnDisconnectedEv for C	CallCtlConnDisconnectedEv for C
CallCtlTermConnDroppedEv for T3(Address C)	CallCtlTermConnDroppedEv for T3(Address C)
CiscoConferenceEndEv	CiscoConferenceEndEv
CallInvalidEv	CallInvalidEv
CallObservationEndedEv	CallObservationEndedEv
Note GC2 - Disconnect events are for Address B1 on Terminal T2	**Note** GC2 - Disconnect events are for Address B2 on Terminal T2

Join on Same Lines	Join Across Lines
Note Application logic is based on common transferControllerAddress and works fine in this case, because commonAddr is present in both final and consult call	**Note** There is no common address for controllers in final and consult call, but the controller TerminalName is same for both the controller addresses. So, application should rely on CommonTerminalName to find out the connections, terminal connections and controllers.

Note In connected Conference Across Lines scenario, apart from the events mentioned, applications can see another temporary call GC3 going active(CallActiveEv) and GC3 goes idle (CallInvalidEv) immediately after the conference is completed.

Join on Same Lines Sample Application Code

```
Handle(CiscoCallEv event)
{
  …..
  …..
  if (event instanceOf  CiscoConferenceStartEv)
  {
    CiscoConferenceStartEv ev  =
      (CiscoConferenceStartEv)event;
    processConference(ev);
  }
}
processConference(CiscoConferenceStartEv ev){
  CiscoAddress controllerAddr  =
    ev.getConferenceControllerAddress();

  CiscoCall[] consultCalls  =  ev.getConferenedCalls();
  CiscoCall GC1  =  ev.getFinalCall();
  CiscoConnection[] movedConns[]  =
    findConnections(consultCalls,  controllerAddr);

  //Additional App logic to clear connections.
}
Connection[] findConnections(CiscoCall[] calls,  CiscoAddress addr){
  ArrayList connList  =  new ArrayList();
  for(x = 0; x < calls.length; x++)
  {
    CiscoConnection[] conns  =
      calls[x].getConnections();
    for (i = 0; i<conns.length; i++)
    {
      if conns[i]
        .getAddress().equals(addr) {
        connList.add(conns[i]);
      }
    }
  }
  return connList.toArray(Connection[] conns);
}
```

Join Across Lines Sample Application Code

```
Handle(CiscoCallEv event)
{
  …..
  …..
  if (event instanceOf  CiscoConferenceStartEv)
  {
    CiscoConferenceEv ev  =
      (CiscoConferenceStartEv)event;
    processConference(ev);
  }
}
processConference(CiscoConferenceStartEv ev){
  String controllerTermName  =
    ev.getControllerTerminalName();
  CiscoCall[] consultCalls  =  ev.getConferenedCalls();
  CiscoCall GC1  =  ev.getFinalCall();
  CiscoConnection[] movedConns  =  findConnections(consultCalls,
                controllerTermName);

  //Additional App logic to clear connections.
}
Connection[] findConnections(CiscoCall calls,  String termName){
  ArrayList connList  =  new ArrayList();
  for(x = 0; x < calls.length; x++)
  {
    CiscoConnection[] conns  =  calls[x].getConnections();
    for (i = 0; i<conns.length; i++)
    {
      CiscoTerminalConnection[] termConns  =
        conns[i].getTerminalConnections();
      for(j = 0; j<termConns.length; j++)
      {
        if(termConns[j].getTerminal().getName.equals(termName)
            && termConns[i].getState() ! =
            TerminalConnection.PASSIVE)
        {
          connList.add(conns[i]);
        }
      }
    }
  }
  return connList.toArray(Connection[] conns);
}
```

Interface Changes

See CiscoConferenceStartEv, on page 376

Message Sequences

See Connected Conference or Join Across Lines Use Cases - New Phones Behavior , on page 1159

Backward Compatibility

This feature is backward compatible.

This feature cannot be turned off for certain devices and Cisco Unified JTAPI always reports events for Join Across Lines for these phones. However, to provide backward compatibility for applications, a new permission to allow controlling these devices and to allow connected conference across lines has been added. A new

standard role Standard CTI Allow Control of Phones supporting Connected Xfer and conf and a standard user group are also added. Applications can control these devices only if this new role is associated to the application user, assuming that application is using JTAPI client 7.1.2 or higher. So, by default these devices are listed as Restricted. The application must upgrade to handle this feature and associate the new permission to control these devices. If the application uses an older JTAPI client the devices are not restricted but if the application tries to observe these devices (which supports this feature to be invoked manually), JTAPI throws an exception and marks these devices as restricted from there on.

Cisco assumes that two or more applications do not control or observe the same terminal or address simultaneously. If they do, all instances of this application make changes to support this feature or coordinate to avoid any problem. Otherwise, application behavior may be unforeseen. For example, if App1 and App2 are two applications controlling or observing the same terminal or address and App1 makes changes to support this feature then App2 is also expected make changes to support the feature. Else, invocation of this feature by App1 on common devices can break App2.

As, the feature is designed to provide an enhanced user experience, Cisco strongly recommends that all Cisco Unified JTAPI applications should evaluate and support this feature and upgrade if necessary with the code logic to handle both the old and new behavior.

Join Across Lines with Conference Enhancements (SCCP and SIP)

Join Across Lines feature supports on CTI-supported SIP phones and SCCP phones. The enhancements are:

- Applications can conference two calls in which each conference is on a different address but on the same terminal.
- Add participants to a conference using a non-controller.

Note You can disable Join Across Lines by turning off the Join Across Lines Policy service parameter. Conference Chaining and the feature that allows Non-Controller adding participant to conference can be disabled by disabling the Advanced Ad Hoc Conference Enabled and Non-linear Ad Hoc Conference Linking Enabled service parameters.

The following behavior occurs when an application issues a conference request, but selected and active calls are not part of the conference request. It also applies for user-selected calls that are not part of the conference request, but become part of the resulting conference:

- The Active Call on a Terminal is always added to the resulting conference when conference is invoked on a call on any address on that terminal. Consider that B1 and B2 addresses exist on the same terminal, then:
  - A --> B1- GC1
  - C --> B1- GC2
  - D --> B2- GC3 (active call)

  The application invokes GC1.conference (GC2) and results in A-B1-C-D in a conference with GC1, although the call with D was not part of the conference request.

An active conference call on a terminal is added to the resulting conference when conference is invoked on a call on any line on that terminal. In this case, the active conference call becomes the surviving final call (provided the application-specified primary call is not a conference call).

In this example, the application specified primary call is cleared after the conference operation. It is possible that the application-specified primary call may not join the resulting conference and in that case the call is not cleared after the conference is complete.

- Consider that the B1 and B2 addresses on the same terminal and conf1 is a conference call with A-B1-C in conference with B1 as the controller, then:
  - B1 --> D – GC1 (on hold)
  - conf1 – GC2 (active call)
  - B2 --> E – GC3 (on hold)

  Application invokes GC1.conference(GC2, GC3). This results in A-B1-C-D-E in conference with GC2 as the surviving call. Although application had specified GC1 to be the primary call, GC1 does not survive after the conference.

The behavior also applies to regular conferencing with a common controller. Consider A, B, C, and D are lines on different terminals, then:

- A --> B - GC1
- C --> - GC2
- D --> - GC3 (active call)

The application requests GC1.conference (GC2). This results in A-B-C-D in conference with GC1. Although a direct call with Dwas not part of the conference request, D joins the conference.

Interface Changes

There are no interface changes. You can use the current interfaces to conference calls on different addresses on the same terminal.

Message Sequences

Join Across Lines with Enhancements, on page 794

Backward Compatibility

This feature is backward compatible.

JRE 1.2 and JRE 1.3 Support Removal

This release of the CiscoJTAPIClient supports only JRE 1.4. There are no interface changes; however, the JRE 1.2 and 1.3 versions are no longer supported. This change is to support QoS, which is available only in the JDK1.4 version (and above). Inaddition, jtapi.jar contains Cisco encryption files that depend on the JRE 1.3 version (and above). This provides a stronger password encryption algorithm when it is sent over TCP to CTIManager. As part of this feature, JTAPI invokes the API provided by IMS (Identity Management System, a Cisco Unified Communications Manager component) to encrypt a password before sending it.

JRE 1.4 also enables Cisco Unified JTAPI to use additional JDK 1.4 APIs. Applications that use previous versions of JRE must install JDK 1.4 to use Cisco Unified JTAPI.

Note There are no interface changes to JTAPI Applications, however JTAPI.jar contains RSA jsafe.jar (3.3) and Apache log4j-1.2.8.jar files. If Applications are using any jar files that are not compatible with these versions of jsafe.jar (Version 3.3) and log4j-1.2.8.jar, then JTAPI or the Application may not work, depending on which one is in the classpath first

As part of this migration, JTAPIPreferences and sample applications dependency on MS-JVM was also removed. Two new configuration parameters were provided on the Advanced tab in the JTAPI Preferences dialog box:

- JTAPI Post Condition Timeout
- Use Progress As Disconnected

Backward compatibility

This feature is not backward compatible.

JTAPI Version Information

In order to connect to Release 5.0 of Cisco Unified Communications Manager Administration, JTAPI clients have to upgrade to the new version of JTAPI bundled with the Cisco Unified Communications Manager Administration Release 5.0. JTAPI version is in the form of 3.0(X.Y), where X and Y depend on the sub-release. Applications cannot connect with prior release of JTAPI.

Locale Infrastructure Development

This feature removes currently supported languages for Cisco Unified JTAPI client install. Cisco Unified JTAPI client install is only supported in English. It also adds the capability to dynamically update the locale in JTAPI Preference application from the Cisco Unified Communications Manager server. JTAPI Preference application will continue to support all the languages that are supported in prior releases. Support for adding new languages and updating locale files is also added.

Before this release, the Cisco Unified JTAPI client install and JTAPI Preferences application were localized during builds and did not add support for new languages or update locales for existing languages. The JTAPI client locale updates were performed in Cisco Unified Communications Manager maintenance releases. This feature adds capability to dynamically update locale file for JTAPI Preferences application, and JTAPI Client install is installable only in English languages.

The JTAPI Client install needs the Cisco Unified Communications Manager TFTP server IPaddress. The TFTP IP address is used for downloading locale files for the preferences application. If the TFTP IP address is not entered or an incorrect IP address is entered, the preference application displays only in English language. Further on, whenever new locale updates are available, JTAPI Preferences application will notify user about available updates and update locale files.

Interface Changes

There are no interface changes.

Message Sequences

Locale Infrastructure Development Scenarios, on page 1078

Backward Compatibility

This feature is backward compatible from the JTAPI Application perspective, but from the JTAPI Client install perspective, currently supported languages have been removed. In this regard, it is not backward compatible.

Logical Partitioning

This feature enables administrators to configure geographic locations and restrict calls that pass through a PSTN gateway to be connected directly to a VoIP phone or VoIP PSTN gateway in another geographic location. This feature allows use of single line analog phones and remains compliant with the Telecom Regulatory Authority of India (TRAI) regulation.

This feature can be turned off by using the Logical Partitioning Enabled service parameter, which is disabled by default.

Interface Changes

See CiscoJtapiException, on page 410

Message Sequences

See Logical Partitioning Feature Use Cases, on page 1227

Backward Compatibility

This feature is backward compatible.

Media Termination at Route Point

This feature enables multiple active calls at the route point, and applications can terminate media for all active calls by specifying the IP address and port number for each call or whenever media is established.

To use this feature, applications must register the route point by supplying media capabilities. When a call gets answered at this route point, CiscoMediaOpenLogicalChannelEv gets sent to the applications. This event gets sent whenever media is established. Applications must react to this event and specify the IP address and port number where they want to terminate media.

A CiscoRouteTerminal represents a special kind of CiscoTerminal that allows applications to terminate RTP media streams. Unlike a CiscoTerminal, a CiscoRouteTerminal does not represent a physical telephony endpoint, which is observable and controllable in a third-party manner. Instead, a CiscoRouteTerminal represents a logical telephony endpoint, which may get associated with any application that intends to route calls and also terminate media. Unlike CiscoMediaTerminal, CiscoRouteTerminal can have multiple active

calls at the same time. Typically, CiscoRouteTerminals get used to place calls in queue until an agent is available to service the caller.

Note Only RoutePoint Terminals appear as CiscoRouteTerminal through JTAPI.

Terminating media comprises a three-step process.

1. The application registers its media capabilities with this terminal by using the CiscoRouteTerminal.register method.
2. An application adds an observer that implements CiscoTerminalObserver interface by using the Terminal.addObserver method.
3. The application must add addCallObserver on CiscoRouteTerminal or on CiscoRouteAddress to receive CiscoCall object from the provider by using CiscoRTPHandle.

Applications receive CiscoMediaOpenLogicalChannelEv for each call and must supply the IP address and port number by using the setRTPParams method on CiscoRouteTerminal.

You must modify applications that are written for the CiscoJtapiClient 1.4(x) release or earlier to register with CiscoRouteTerminal. NO_MEDIA_TERMINATION if the applications are not interested in media termination.

Multiple applications can register with the same route point as long as they are registered with the same media capabilities and registrationType. All applications, if they have registered with CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION and then add a terminal observer, receive CiscoMediaOpenLogicalChannelEv, but only one application can invoke setRTPParams.

Note Applications that terminate media must use the CallControl package for answering and redirecting calls. Applications that only route calls can use a routing package.

Note Applications should be aware that, if any features are performed before reacting to CiscoMediaOpenLogicalChannelEv, the features may fail. If applications do not respond to these events in the time that is specified in the Media Exchange Timeout parameter in the Cisco Unified Communications Manager Administration windows, the call may fail.

The following new or changed interfaces exists for Media Termination at Route Point:

Interface CiscoRouteTerminal Extends CiscoTerminal

boolean	isRegistered() If the CiscoMediaTerminal gets registered, this method returns true. Otherwise, it specifies false.
boolean	isRegisteredByThisApp() If the application issues a successful registration request, this method returns true and remains true until the application unregisters the device. This remains valid even if the device is out of service because of CTIManager failure.

void	**register (CiscoMediaCapability[] capabilities, intregistrationType)** The CiscoRouteTerminal must exist in the CiscoTerminal.UNREGISTERED state, and the provider must exist in the Provider.IN_SERVICE state.
void	**setRTPParams (CiscoRTPHandle rtphandle, CiscoRTPParams rtpParams)** Applications set the ipAddress and the RTP port number to dynamically stream media for a call.
void	Unregister() Ensure the CiscoRouteTerminal is registered, and the provider is in the Provider.IN_SERVICE state.

Interface CiscoMediaOpenLogicalChannelEv Extends CiscoTermEv

int	**getpacketSize ()** Returns the packet size of the far end in milliseconds.
int	**getPayLoadType ()** Returns the payload format of the far end, one of the following constants:
CiscoRTPHandle	getCiscoRTPHandle () Returns the CiscoTerminalConnection object on which applications must invoke the setRTPParams request.

Interface CiscoRTPHandle

int	getHandle() Returns an integer representation of this object, currently the Cisco Unified Communications Manager CallLeg ID.

CiscoProvider

CiscoCall	getCall (CiscoRTPHandle rtpHandle) Returns the call object with the rtpHandle that is associated with a specific terminal. If no callobserver gets added to the terminal at the time when the applications receive CiscoRTPHandle in CallOpenLogicalChannelEv, CiscoCall may register null.

For details on these interfaces, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for media termination at route point, see Message Sequence Charts, on page 755

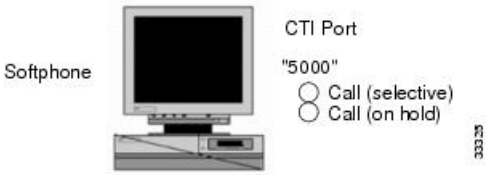
Media Termination Extensions

The media termination feature allows applications to transmit and capture the bearer of a call, for example, audio or video. This action sometimes gets referred to as “rendering and recording” or “sourcing and sinking” media. It remains distinct from call control because media termination concerns the data that flows between endpoints in a call, not the details of setting up or tearing down calls. For example, an automatic call distributor (ACD) uses call control to route calls among available agents but does not terminate media. An interactive voice response (IVR) application, on the other hand, uses call control to answer and disconnect calls and uses media termination to play sound files to callers.

Although no telephony applications are solely interested in media termination, this feature always gets used in combination with call control. JTAPI 1.2 primarily represents a call control specification and offers very limited support for applications that require media termination. Because the Cisco Unified Communications Solutions platform supports media termination to a much greater degree than JTAPI standard, the Cisco Unified JTAPI implementation extends JTAPI to add full support for this feature.

In Cisco Unified JTAPI, software-based media termination occurs by using Computer Telephony Integration (CTI) ports. They include one or more lines (dialable numbers) that can be used to originate or receive calls. They however need a controlling application to provide the source and sink of the media. An application registers its interest in the media termination port with the Cisco Unified Communications Manager. The Cisco Unified Communications Manager then delivers all the events that relate this virtual device to the application. InCisco Unified JTAPI, CTI ports get referred to as CiscoMediaTerminals. The following figure shows the CTI port configuration. For details about administering and configuring a CTI port, refer to the Cisco Unified Communications Manager Administration information.

Figure 8: CTI Port Diagram



To implement a softphone application (where the PC acts as the telephone set, for example), the Cisco Unified JTAPI application would manage a CTI port.

Message Waiting Indicator Enhancement

The Enhanced Message Waiting Indicator (MWI) feature enables applications to provide the following message counts to be displayed on phones that support the enhanced message waiting counts:

- Total number of new voice messages (includes normal and high priority messages)
- Total number of old voice messages (includes normal and high priority messages)
- Number of new high priority voice messages
- Number of old high priority voice messages
- Total number of new fax messages (includes normal and high priority messages)
- Total number of old fax messages (includes normal and high priority messages)

- Number of new high priority fax messages
- Number of old high priority fax messages

Two new APIs are added as CiscoAddress JTAPI extensions to provide the enhanced MWI message summary information. Similar to the existing setMessageWaiting APIs, one of the APIs allows summary information to be set up for the observed address. The other API allows message summary information to be set up on any address that is reachable on the observed address, as defined by the configured calling search space of the observed address.

These new APIs can also be used on phone types that do not support the enhanced message counts. If used on non-supported phones, these APIs behave similar to the existing setMessageWaiting method, that is, only the messaging waiting indicator lamp is turned on or off and counts are not displayed.

Interface Changes

See Related Documentation, on page 283

Message Sequences

See Enhanced MWI Use Cases, on page 1160

Backward Compatibility

This feature is backward compatible. The existing setMessageWaiting APIs will not be modified. Applications that do not want to use the new enhanced MWI feature can continue to use these APIs for setting the MWI lamp.

Modifying Calling Number

This feature enables applications to modify the calling party DN in the select route API from the route point. Applications may pass an array of modifying calling numbers in the selectRoute API and an array length of modifying calling numbers may equal the length of the route that is selected. If no modifying calling number element is present for a corresponding routeSelected index or if the element is null, then no modifying calling number gets set for that route selected element.

Two new interfaces getModifiedCallingAddress () and getModifiedCalledAddress () are exposed on the call object, which returns modified calling or called number. If no modification occurs, these interfaces may return the same values as getCurrentCallingAddress () and getCurrentCalledAddress () interfaces. If an application is only controlling the route point and modifies the calling number by using selectRoute API, it may not get modified calling address in the getModifiedCallingAddress interface. If an application is controlling any calling or called parties, it may get correct values after it receives call control events after the calling number is modified.

A new interface, getRouteSelectedIndex (), gets exposed on the new class CiscoRouteUsedEvent, an extension of RouteUsedEvent, which gives the index of the selected route. Applications need to cast the RouteUsedEvent to the CiscoRouteUsedEvent to get access to this method.

Example

```
routeSelected[0]  =  133555
routeSelected[1]  =  144911
routeSelected[2]  =  143911
```

```
routeSelected[3]  =  5005

modifiedCallingNumber[0]  = null
modifiedCallingNumber[1]  = 9721234567
modifiedCallingNumber[2]  = 9721234568
modifiedCallingNumber[3]  = null
```

If routeSelected[0] or routeSelected[3] is selected for routing, the modifying calling number may not get applied.

You can only use this feature after an administrator enables the modifying calling number check box in the Cisco Unified Communications Manager Administration for a particular user, which by default is False. If it is not configured, a RerouteEvent with the cause of RouteSession.CAUSE_PARAMETER_NOT_SUPPORTED gets sent to the applications. The application that is modifying the calling number needs to be aware that display name on the called party is affected, and subsequent feature interactions of the calling or called party may result in inconsistent behavior.

The following new or changed interfaces exist for Modifying Calling Number:

CiscoRouteSession

void	`selectRoute (java.lang.String[] routeSelected, intcallingSearchSpace, String[] modifiedCallingNumber)` This interface allows applications to modify the calling party number to the routeSelected address. If no modifiedCallingNumber element exists for the corresponding routeSelected element, the calling number does not get modified if a call gets routed to that particular routeSelected element.

CiscoCall

javax.telephony.Address	`getModifiedCalledAddress ()` This interface returns a modified called address for the call if an application modifies the calling party by using the selectRoute API; however, this information may not be accurate if an application is only controlling the route point that modifies the calling number. If no modified calling number gets performed, this acts similar to the getCurrentCalledAddress interface. Typically, this gets varied from getCurrentCalledAddress when a feature gets invoked after modified calling number modifications.
javax.telephony.Address	`getModifiedCallingAddress ()` This interface returns a modified calling address for the call if an application modifies the calling party by using the selectRoute API; however, this information may not be accurate if an application is only controlling the route point that modifies the calling number. If no modified calling number gets performed, this interface acts similar to the getCurrentCallingAddress interface.

CiscoRouteUsedEvent

int	`getRouteSelectedIndex()` This method returns an array index of the route to where the call gets routed.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for Modifying Calling Number, see Message Sequence Charts, on page 755.

Multi-fork Recording using CUBE Media Proxy Server

Prior to Cisco Unified Communications Manager, Release 12.5(1), the Unified Communications Manager supported only single recorder for a call. With the Cisco Unified Communications Manager, Release 12.5(1), the Unified Communications Manager supports Multi-forking for Call Recording feature.

The Unified Communications Manager is connected to CUBE Media Proxy server which is connected to multiple recorders. The JTAPI interface is enhanced to get the details of multiple recorders in case of Multi-Forking recording through CUBE Media Proxy server.

Backward Compatibility

This feature is backward compatible. JTAPI supports the current APIs.

Multilevel Precedence and Preemption Support

Cisco Unified Communications Manager enables the use of supplementary services by phones that are configured for Multilevel Precedence and Preemption (MLPP). Cisco Unified Communications Manager does this by maintaining the precedence level for calls.

Note JTAPI does not provide the precedence level of applications.

Multiple Calls Per DN

Multiple calls per DN represent the ability to support multiple calls on a line (DN) and the features operation on these calls. Prior to Cisco Unified Communications ManagerRelease4.0(1), the system supported a maximum of only two calls. Cisco JTAPI now supports multiple calls per line, which allows multiple calls on the same line and feature operation on that line.

No interface or message flow changes occurred for Multiple Calls Per DN.

Native Queuing

It is very common in a Cisco Unified CM deployment that a hunt pilot has more calls distributed through the call distribution feature than its hunt members can handle at any given time. Native Queuing feature holds the calls in a queue until they are answered. When a hunt member is available, the call is removed from the queue and offerred to the hunt member.

To enable this feature, the Cisco Unified CM administrator needs to enable the check box **Queue Calls** in **Queuing** section of the **Hunt Pilot configuration** page. Following settings are available under Native Queuing feature configuration:

- **Maximum Number of Callers Allowed in Queue (1–100):** This is the queue depth configuration and reflects the maximum number calls that can be in the queue at any point of time.
  - **Destination When Queue is Full:** User Configurable Destination number to which the calls are forwarded when the Maximum Number of callers allowed in queue limit is reached.
  - **Disconnect**: This option results in the call getting rejected and dropped when the Maximum Number of callers allowed in queue limit is reached.
- **Maximum Wait Time in Queue (10–3600 seconds):** User configurable Maximum wait time a call can be in the queue.
  - **Destination When Maximum Wait Time is met:** User Configurable destination DN to which the call is forwarded when the maximum wait time in queue is reached.
  - **Disconnect:** This option results in the call getting rejected and dropped when the the maximum wait time in queue is reached.
- **When There Are No Hunt Members Logged In or Registered:** User configurable destination DN to which the queue feature forwards the calls when none of the hunt members in the HuntPilot are registered or logged in.
  - **Disconnect:** This option results in the call getting rejected and dropped when there are no hunt members available for the dialed hunt pilot.
  - **Destination:** User Configurable destination DN to which the call is forwarded when no hunt members available for the dialed hunt pilot.

If a caller calls a hunt pilot with all its members busy, a CiscoHuntConnection will be created temporarily. Then, when this feature is enabled, the hunt connection will drop and a new connection will be created which will have the address name same as that of the hunt pilot and the address type will be CiscoAddress.INTERNAL. This new connection will be moved to CallControlConnection.QUEUED state and it will remain in this state until the call gets dequeued or dropped.

Cisco JTAPI exposes the following new reasons:

- CiscoFeatureReason.REASON_QUEUING
- CiscoFeatureReason.REASON_DEQUEUING
- CiscoFeatureReason.REASON_DEQUEUING_TIMER_EXPIRED
- CiscoFeatureReason.REASON_DEQUEUING_AGENTS_BUSY
- CiscoFeatureReason.REASON_DEQUEUING_AGENTS_UNAVAILABLE

The above reasons indicate when a call gets enqueued and dequeued respectively because of the various configurations on the Hunt Pilot.

Note The behavior is different when conferencing a queued calls. If a caller which is in a queue conferences the call with another party, then the queued connection is dropped and a new connection is created with the address of the hunt pilot number and the address type is CiscoAddress.UNKNOWN, and it will be moved to CallControlConnection.ESTABLISHED state.

If a call is removed from the queue, for example when an agent becomes free, the agent will be added to the conference and a connection will be created for it. For the Hunt Pilot, JTAPI creates a normal connection instead of a CiscoHuntConnection. This is a limitation of JTAPI in handling a conference with Hunt Pilot.

In a case where only the Hunt member is observed, there will be no issues and JTAPI will be able to handle it.

Interface Changes

See CiscoFeatureReason, on page 402

Message Sequences

See Native Queuing, on page 1281.

Backward Compatibility

This feature is backward compatible. Check the **Queue Calls** checkbox in the Hunt Pilot Configuration window to enable this feature. By default this feature is disabled.

Network Alerting

In earlier releases of CiscoJTAPI (CiscoJTAPI versions 1.4(x.y)), when a call was made to an address outside of the cluster, CallCtlConnNetworkReachedEv and CallCtlConnNetworkAlertingEv events were delivered to the farend address.

In later versions of Cisco Unified Communications Manager (4.0 and above) and Cisco Unified JTAPI (2.0), these events were not delivered. In these versions CallCtlConnection for the farend address went to the ESTABLISHED state from the OFFERED state. The previous versions of Cisco Unified JTAPI delivered CallCtlConnOfferedEv, CallCtlConnEstablishedEv for the farend address when a call was made across a gateway with “overlap sending” turned off. CallCtlConnNetworkReachedEv and CallCtlConnNetworkAlertingEv events were not delivered to the application.

In Cisco Unified Communications Manager4.0 and 4.1, the “Allow overlap sending” flag on the route pattern configured for the gateway or the “AllowNetworkEventsAfterOffered” parameter in jtapi.ini needed to be turned on to receive network events.

In Cisco Unified Communications ManagerRelease 5.0, if the “Allow overlap sending” flag is enabled, an application sees ConnCreatedEv, CallCtlConnNetworkReachedEv, CallCtlConnNetworkAlertingEv, and CallCtlConnEstablishedEv for the farend address for calls across a gateway.

If the “Allow overlap sending” flag is not enabled, an application sees ConnCreatedEv, CallCtlConnOfferedEv, CallCtlConnNetworkReachedEv, CallCtlConnNetworkAlertingEv, and CallCtlConnEstablishedEv for the farend address for calls across a gateway.

Note AllowNetworkEventsAfterOffered is not available in Cisco Unified Communications Manager Release5.0. The above events are delivered regardless of the jtapi.ini parameter setting.

Backward Compatibility

This feature is not backward compatible.

Network Events

In previous releases of Cisco Unified JTAPI, when a call is made to an address outside the cluster, CallCtlConnNetworkReachedEv and CallCtlConnNetworkAlertingEv events are delivered for the far-end address.

In Cisco Unified Communications Manager 4.0 and later, these events do not get delivered. In these versions CallCtlConnection for the far-end address goes to the ESTABLISHED state from OFFERED state. The application will receive CallCtlConnOfferedEv, CallCtlConnEstablishedEv for the far-end address. The CallCtlConnNetworkReachedEv and CallCtlConnNetworkAlertingEv events do not get delivered to the application. To receive network events, the "Allow overlap sending" flag on the route pattern that is configured for the gateway must be turned on.

A new jtapi.ini parameter, AllowNetworkEventsAfterOffered, that is introduced allow the application to control the delivery of these events. Applications that need the network events but cannot turn on this flag can use this new jtapi.ini parameter to receive network events for outgoing calls.

To turn on the parameter, complete the following steps:

Step 1 Run jtprefs and select the required options. This creates jtapi.ini file in c:\winnt\java\lib, if Cisco Unified JTAPI is installed in the default directory. If the jtapi.ini file already exists, you can update the file directly without running jtprefs.

Step 2 Add AllowNetworkEventsAfterOffered = 1 to the end of the file and save it.

Step 3 Repeat the preceding step every time Cisco Unified JTAPI is reinstalled.

When the AllowNetworkEventsAfterOffered flag is enabled, the application will receive CallCtlConnOfferedEv, CallCtlConnNetworkReachedEv or CallCtlConnNetworkAlertingEv and CallCtlConnEstablishedEv for the far-end address.

New Error Code in CiscoTermRegistrationFailedEv

This event is sent to application when TerminalRegistration fails for some reason. The return value of getErrorCode() interface indicates the type of failure. On receiving this event, application should try to reregister the Terminal. In this version a new return value is added to this interface. CiscoTermRegistraionFailedEv.UNKNOWN is introduced in this version to handle unknown failures.

Backward Compatibility

This feature is backward compatible.

Noncontroller Adding of Parties to Conferences

Any party in a conference can now add participants into the conference. In previous releases, only the conference controller could add participants.

- CiscoConferenceStartEv, on page 376 contains an identifier for the requestor party.
- The method getConferenceControllerAddress returns the terminal connection of the requestor.
- The new method getOriginalConferenceControllerAddress() for CiscoConferenceStartEv, on page 376 returns the terminal connection of the original controller.

Park DN Monitor

Cisco Unified JTAPI applications can register to receive events when calls are parked and unparked. CiscoProvCallParkEv events will be delivered to provider observer when the application registers for this feature. To successfully register for this feature, ensure that the “call park retrieval allowed” flag for the user is turned on. You can access this flag with the user configuration on Cisco Unified Communications Manager Administration. After registering for this feature, the application will receive CiscoProvCallParkEv events whenever a call is parked or unparked from any device in the cluster.

The following new interfaces allow applications to register and unregister for this feature:

```
public interface CiscoProvider {
public void registerFeature ( int featureID ) throws
     InvalidStateException,  PrivilegeViolationException;
public void unregisterFeature ( int featureID ) throws
     InvalidStateException;
}
```

The *featureID* is CiscoProvFeatureID.MONITOR_CALLPARK_DN.

Park Monitoring and Assisted DPark Support

This feature provides a new park reversion behavior to applications invoking park request. Currently, when the park reversion timer expires, the call is reverted to the address of the parker. With the new behavior, the call remains parked at the park DN, even as the Park Monitoring reversion timer expires.

This feature also enables status monitoring of the parked call at the address of the parker. After a call is parked using the existing CiscoConnection.park() JTAPI API on newer phones or directly from the phone itself, Cisco Unified JTAPI delivers a new event CiscoAddrParkStatusEv, which includes the current status of the parked call. The application must then add AddressObserver on the address of the parker, and enable a filter to receive this event. If application adds an observer after the call is parked, then the events are delivered with CAUSE_SNAPSHOT. The park status in the new event can be one of the following:

- Parked—Indicates a call was parked by the user of the application.

- Reminder—Indicates the park monitoring reversion timer for the parked call has expired.
- Retrieved—Indicates a previously parked call was retrieved.
- Abandoned—Indicates a previously parked call is disconnected while waiting to be retrieved.
- Forwarded: indicates the parked call has been forwarded to the configured Park Monitoring Forwarded No Retrieve destination, as the Park Monitoring Forward-No-retrieve timer has expired.

When the cause is CAUSE_SNAPSHOT the park status can be either Parked or Reminder state only.

On the phone, these notifications are targeted, that is, only the device parking the call can see these notifications (devices sharing line with the parker's device does not receive similar notifications). In Cisco Unified JTAPI, getTerminal() interface on CiscoAddrParkStatusEv has been added tomanage this. This returns the terminal on whose address, these notifications were received and this is the terminal that parked the call.

Cisco Unified JTAPI also provides the CiscoCallID to applications in this new event. Applications may use this to retrieve the call object. However CiscoCallID.getCall() may return null value if the call does not exist in the provider's domain at the time this event is received.

Cisco Unified JTAPI provides a new interface CiscoAddrEvFilter to control or filter the new event notifications to applications. Applications may get or set the filter value through the APIs getCiscoAddrParkStatusEvFilter() and setCiscoAddrParkStatusEvFilter() on the CiscoAddrEvFilter interface. Two new methods, getFilter() and setFilter(), have also been provided in the CiscoAddress to get and set the values of the filters in the CiscoAddrEvFilter interface. Applications receive the new event notification CiscoAddrParkStatusEv only if the filter is enabled and the setFilter() is invoked on CiscoAddress. By default, the filter value for CiscoAddrParkStatusEvFilter is false to maintain backward compatibility.

When a call is parked, the Park monitoring reversion timer starts and then expires. After this, Park Monitoring Forward No Retrieve timer starts. When this timer expires, and the Forward No Retrieve destination is configured, the call is forwarded to this destination. A new CiscoFeatureReason FORWARD_NO_RETRIEVE is delivered in the connection events, when connections are created at the forwarded destination. If the Forward No Retrieve destination is not configured, call is forwarded back to the parker's DN, with the same reason as when park reversion occurs (CiscoFeatureReason.PARKREMINDER).

When application invokes CiscoAddress.getAddressCallInfo(Terminal term), the CiscAddressCallInfo which is returned is now enhanced to include number of parked calls. This returns the number of parked calls. Cisco Unified IP Phone 7900 Series with SIP/SCCP returns zero value even if there are calls parked by this address.

This feature is applicable only when newer phones park the call. If Cisco Unified IP Phone 7900 Series with SIP/ SCCP, parks the call, user continues to see the existing behavior. So, if a Cisco Unified IP Phone parks the call and is sharing a line with a Cisco Unified IP Phone 7900 Series with SIP, the new Park Monitoring enhancements can be seen. However, if the Cisco Unified IP Phone 7900 Series with SIP or SCCP invoked park, the old Park behavior would be seen on all the phones, if application is monitoring any of these lines.

Users can set the Park Monitoring Reversion Timer to zero and set the Park Monitoring Forward No Retrieve Destination to the existing Park Reversion Duration timer to get the old behavior on the Cisco Unified IP Phone (provided the Forward No Retrieve destination is not configured) if the user so desires. However, the event notification cannot be controlled.

On Cisco Unified Communications Manager Service Parameter pages, the timers mentioned above can be configured. These would apply only for SIP versions of future models of Cisco Unified IP Phone .

Park Monitoring Reversion timer: This timer is started as soon as the call is parked. This is the amount of time that a call remains parked before the user is reminded that there is a parked call. The range is 0-1200 seconds, with default value of 60 seconds.

Park Monitoring Periodic reversion timer: The frequency in which the user is reminded about the parked call. The range is 0-1200 seconds, with default value of 30 seconds.

Park Monitoring Forward No Retrieve timer: This timer is started when the park monitoring reversion timer expires. This is how long, in seconds, the park reminder notification plays before the parkee is redirected to the parker's Park Monitoring Forward No Retrieve (FNR) destination. The range is 30-1200 seconds, with default value of 300 seconds.

Park Monitoring Forward No Retrieve Destination is configurable on the line page in Cisco Unified Communications Manager Line page settings.

Assisted DPark provides an alternative one step way to perform DPark operation on phones. When user performs Assisted DPark from newer phones and application is monitoring the parked party, Cisco Unified JTAPI provides reason CiscoFeatureReason.REASON_REFER in the connection events (ConnCreatedEv, ConnInProgressEv and CallCtlConnQueuedEv) for DPark DN. Currently when DPark is done, application gets connection events with CiscoFeatureReason.REASON_TRANSFER.

Interface Changes

See CiscoAddrParkStatusEv, on page 310

Message Sequences

See Park Monitoring Support, on page 1200

Backward Compatibility

Park Monitoring enhancements and Assisted DPark support are backward compatible.

The new park reversion behavior improves the user experience to allow the parked call to be retrievable for as long as possible. It also improves the usability of the park feature by allowing the user to monitor the status of a parked call through the new event being delivered.

Applications can conditionally enable/disable filter to receive event via setCiscoAddrParkStatusEvFilter() API on CiscoAddeEvFilter. By default this filter is disabled and therefore maintains backward compatibility.

If the application uses a JTAPI client older than 7.1.2, the devices are not restricted but if the application tries to observe these devices (which supports this feature to be invoked manually), JTAPI throws an exception and marks these devices as restricted from there on.

Park Reminder

When a parked call is not retrieved for a specified time, a reminder call returns to the address that parked the call, and Park Number connection moves to the Disconnected state. The call reconnects and moves to the Established state. A terminal connection in Talking state gets created for the address that parked the call.

Park Retrieval

When a call is parked from an IP phone, the park number displays on the phone. Any terminal can unpark the call by dialing the park number. When a call is unparked, a new call gets created with connections to unparked address. The CallControlConnection for the park number in the original call, which is in the Queued state, moves to the Disconnected state.

Partition Support

Prior to Cisco Unified Communications Manager Release 5.0, JTAPI did not support partitions. JTAPI considered addresses with the same DN, but different partitions, as same address. It created only one Address object for such cases because addresses are identified only by their DN and not by their partition information.

Beginning with Release 5.0, JTAPI supports addresses that have the same DN but belong to different partitions and treats them as different addresses. Partition information of the addresses is exposed to applications through the methods specified below. Applications that want to make use of this partition support feature must use the API provided to them through JTAPI interfaces and use the address objects accordingly.

This feature is backward compatible. JTAPI supports the current APIs that are used to open and access address objects.

In Cisco Unified Communications Manager Release 5.0, JTAPI is partition aware, and the following configurations are supported.

- Addresses with the same DN, in the same partition, and in different devices get treated as shared lines.
- The system does not allow addresses with the same DN, in the same partition and in the same device.
- Addresses with the same DN, in different partitions, and in the same device get treated as different addresses. Two address objects get created for this scenario, and the application can distinguish between the two by calling the getPartition() API on the address objects.
- Addresses with the same DN, in different partitions, and in different devices get treated as different addresses. Two address objects get created for this scenario and the application can distinguish between the two by calling the getPartition() API on the address objects.

Partition support changes in JTAPI are confined to the address objects and do not affect any other functions or classes of JTAPI. The following sections specify the interface changes.

CiscoAddress Interface

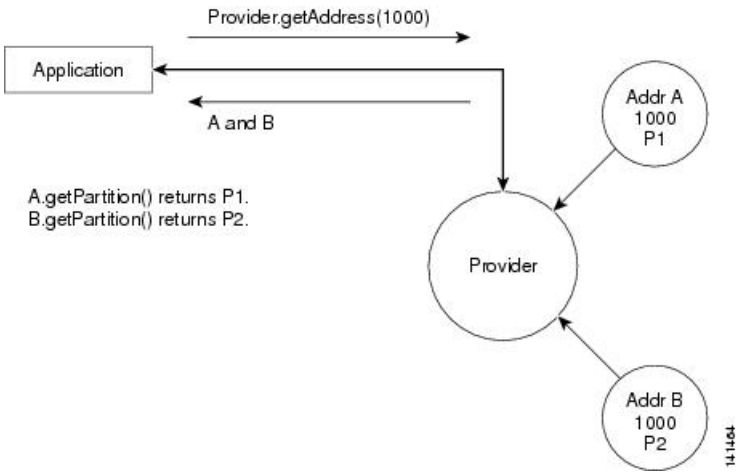
A new method is provided in this class with the following signature.

```
string getPartition ()
```

Returns the partition string of the address object. Applications need to use this method to get the partition information. JTAPI uses this partition information to distinguish between addresses that have the same DN but belong to different partitions and sends the partition information to open the specific addresses.

For example, a provider open returns two addresses, A(1000, P1) and B (1000, P2), where A and B denote the address objects, 1000 denotes the DN of the address objects, and P1, P2 indicate the partitions to which the addresses belong.

Figure 9: Provider Open Returns Two Addresses



When the user invokes A.getPartition (), P1 gets returned while B.getPartition () returns P2.

The provider.getAddresses() method returns multiple addresses in which the Address objects have the same DN but different partition information. An Application can use this method to distinguish between two Address objects that have the same DN but belong to different partitions.

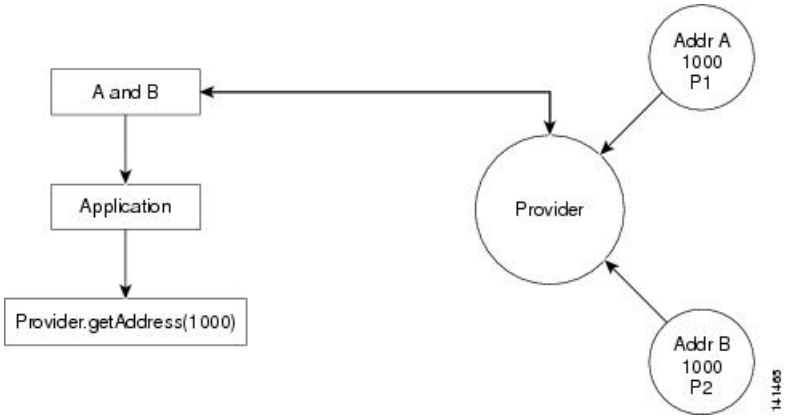
CiscoProvider Interface

The CiscoProvider interface provides the following methods:

Address[]	`getAddress(String number)` Returns an array of Address objects that corresponds to the number and different partitions.
Address	`getAddress(String number, String partition)` Returns the Address object that has the same DN as the number parameter and belongs to the same partition as specified by the partition parameter.

If two addresses A(1000, P1) and B(1000, P2) exist, where A and B denote the address objects, 1000 denotes the DN of the address objects, and P1, P2 indicate the partitions to which the addresses belong, when an application calls provider.getAddress("1000"), it gets two address objects, A and B.

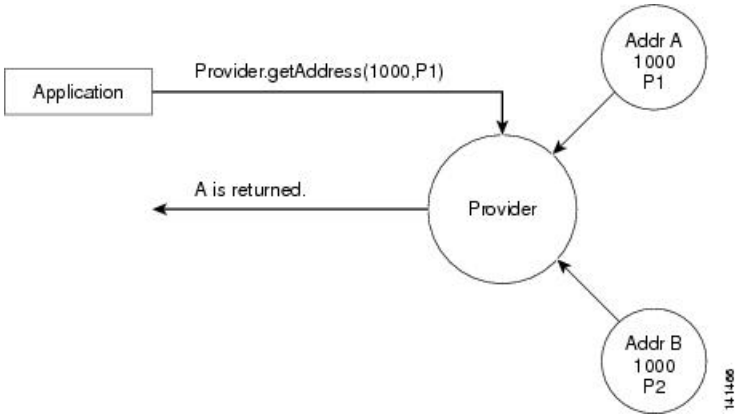
Figure 10: provider.GetAddress() Returns Two Address Objects



When the application calls A.getPartition(), it returns P1, B.getPartition() returns P2, and so on. An Application can distinguish between the two address objects that are using the getPartition method.

Consider the case where the application calls provider.getAddress(1000, P1). In this case, the application specifically looks for the address object whose DN is 1000 and partition is P1. In this case, “A” gets returned by the provider object.

Figure 11: Provider Calls a Specific Address and Partition



CiscoProvCallParkEv Event

CiscoProvCallParkEv provides the following methods in this interface.

`string getParkingPartyPartition()`

Returns the partition string of the parking party.

`string getParkedPartyPartition()`

Returns the partition string of the parked party.

`string getParkPartyPartition()`

Returns the partition string of the park DN.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message sequences for partitions support, see Message Sequence Charts, on page 755

Password Expiry

The administrator can use the CUCM Admin Panel to configure options for login credentials. The password expiry configuration allows the administrator to specify the following two parameters:

1. The time before the password expires (in days) and
2. The number of days before the end of the password expiry to alert the user to change the password.

If a password is expired, JTAPI delivers an exception to the application. In a situation where a password is going to expire soon, JTAPI delivers a new event to the application. JTAPI does not allow applications to modify any of these values, it only reports the information.

Interface Changes

CiscoProvAuthenticationInfoEv, on page 54; CiscoJtapiExceptions, on page 53

Message Sequences

There are no message sequences.

Backward Compatibility

This feature is backward compatible.

Persistent Connection

Persistent Connection is an extension Cisco Extend and Connect feature that was implemented in Unified Communications Manager Release 9.1. A persistent call refers to a call between the Unified Communications manager (CTI Remote Device) and a remote destination that stays up even after calls to it are dropped. JTAPI APIs and error codes were added.

JTAPI supports a new API, CiscoAddress.createPersistentCall(), which allows applications to create persistent calls. At least one remote destination must be configured and the active remote destination must be set. There can be only one persistent call per remote device. Persistent calls cannot be created if there is already a call on the remote device; otherwise, the application receives CiscoJtapiException.OPERATION_NOT_AVAILABLE_IN_CURRENT_STATE. Furthermore, no feature invocations are allowed on or involving persistent calls (park, hold, conference, and transfer).

Two new JTAPI APIs return information about the persistent call. The CiscoAddress.getPersistentConnection() API returns the connection object that is associated to the persistent call. It returns null if no persistent call exists. This API also allows you to check if an address has a persistent connection created on it and from there you can get the call object. The other newly added API is CiscoCallisPersistentCall(), which returns true if the call is a persistent call and false if the call is a normal call.

Existing JTAPI APIs such as Provider.getCalls(), Address.getConnections(), and Terminal.getTerminalConnections() return only the information for normal calls and do not return anything for the persistent call. Provider.getCalls() returns all the calls that are associated with the provider, excluding the persistent calls. Address.getConnections() returns all the connection objects that are associated with this address, excluding the connection for the persistent call. Terminal.getTerminalConnections() returns all the terminal connection objects that are associated with this device, excluding the terminal connection for the

persistent call. This functionality helps with backward compatibility so applications do not need to make any changes to their current implementations.

No new APIs are added to disconnect the persistent calls. Existing Call.drop() and Connection.disconnect() JTAPI APIs can be used to disconnect or drop the persistent calls. Persistent calls cannot be dropped if there is an active call to the remote device. Persistent calls can also be dropped in any of the following scenarios:

- The call is dropped by the remote destination (the remote destination hangs up).
- The remote destination is no longer active. If there is an active call, as soon as that call is over, the persistent call will drop.

After they are created, persistent calls remain connected until the maximum call duration timer expires in which case the call will be cleared.

Some of the new JTAPI Error Codes introduced as part of this feature include the following:

- CiscoJtapiException.CTIERR_CREATE_PERSISTENT_CALL_FAILED: Indicates that there is an issue with creating a persistent call.
- CiscoJtapiException.CTIERR_PERSISTENT_CALL_EXISTS: Indicates that a persistent call already exists.
- CiscoJtapiException.CTIERR_OPERATION_NOT_ALLOWED_ON_PERSISTENT_CALL: Indicates that the specified operation is not allowed on a persistent call.
- CiscoJtapiException.CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_CALL_ACTIVE: Indicates that the request to disconnect the persistent call failed because there is an active customer call. Only when there are no active calls can the persistent call be disconnected.
- CiscoJtapiException.CTIERR_PERSISTENT_CALL_BEING_SETUP: Indicates that the request failed because a persistent call is already being set up.

Backward Compatibility

This feature is backward compatible and existing applications are not affected by this feature.

Interface CiscoAddress Changes

CiscoAddress is enhanced with the addition of new APIs to create a persistent call and to retrieve the connection object that is associated to the persistent call.

CiscoCall	**`createPersistentCall (Terminal terminal, String callerIDNumber, String callerIDName)`** This interface creates a persistent call for this address and will return the call object for the newly created call. Note that CiscoProvider and the address must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. This API cannot be invoked on external addresses. Doing so will result in MethodNotSupportedException to be thrown. If while trying to allocate a globalCallId for the persistent call and an error occurs, ResourceUnavailableException will be thrown. All other errors encountered will result in PlatformException to be thrown.
Connection	**`getPersistentConnection (Terminal terminal)`** This interface will return the connection object that is associated with the persistent call. It returns null if there is no persistent call. This API cannot be invoked on external addresses. Doing so will result in MethodNotSupportedException to be thrown.

Interface CiscoCall Changes

CiscoCall represents a call in the JTAPI model. This interface is enhanced with the addition of a new API.

boolean	**isPersistentCall ()** This interface returns true if the call is a persistent call and false otherwise (if it is a normal call).

Interface CiscoJtapiException Changes

CiscoJtapiException contains all of the error codes that can be delivered by JTAPI to applications. This interface is enhanced with the addition of new error codes.

public static final int

- CTIERR_CREATE_PERSISTENT_CALL_FAILED = "Failed to create Persistent Call." (0x8CCC0132)
- CTIERR_PERSISTENT_CALL_EXISTS = "Persistent Call exists." (0x8CCC0133)
- CTIERR_OPERATION_NOT_ALLOWED_ON_PERSISTENT_CALL = "Operation is not allowed on a Persistent Call." (0x8CCC0134)
- CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_CALL_ACTIVE = "Disconnect persistent call failing, there are active calls." (0x8CCC0136)
- CTIERR_PERSISTENT_CALL_BEING_SETUP = "Persistent Call is being set up." (0x8CCC0139)

Play Zip Tone

The Play Zip Tone feature allows Cisco JTAPI application to play zip tones on active calls. The application specifies the type and the direction of the tone.

Zip tones are played at local or remote end of the call. They are audible and played only for IP phones. These tones are not played if the remote side is a trunk, conference or Cisco Media Terminal or Route Terminal.

The following tones can be played:

- CiscoTone.ZIPZIP
- CiscoTone.ZIP
- CiscoTone.CALLWAITINGTONE

Sample Code

```
Void playTone(TerminalConnection termConn,  int tone,  int direction){
      If ( termConn  ! =  null ){
try {
((CiscoTerminalConnection)termConn).playTone(tone,  direction);
} catch (Exception e){
System.out.println("Exception for playtone request " + e);
}
```

Interface Changes

See CiscoTerminalConnection, on page 630, CiscoTone, on page 652

Message Sequences

See Play Zip Tone, on page 1384

Backward Compatibility

This feature is backward compatible.

Presentation Indicator for Calls

The presentation indicator (PI) on a call provides the application with the ability to hide or reveal Calling/Called/CurrentCalling/CurrentCalled/LastRedirecting parties name and number to the end user. JTAPI provides functions on CiscoCall to get PI value for the party. Use this PI info to present the parties information to the end user. These functions return a value of true or false. A value of "True" indicates that presentation in "Allowed, " and a value of "False" indicates the presentation is "Restricted."

For a conference call, the interfaces on CiscoCall do not return a correct value. Applications must iterate through all the connections in the call to get the PI value that is associated with the address for which the connection gets created. The interface that is provided on CiscoConnection is getAddressPI().

The following new interfaces exist on CiscoCall retrieve PI values.

CiscoCall

boolean	`getCalledAddressPI()` Returns the PI that is associated with getCalledAddressPI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.
boolean	`getCallingAddressPI()` Returns the PI that is associated with getCallingAddressPI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.
boolean	`getCurrentCalledAddressPI()` Returns the PI that is associated with CurrentCalledAddressPI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.
boolean	**`getCurrentCalledDisplayNamePI()`** Returns the PI that is associated with CurrentCalledDisplayNamePI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.
boolean	`getCurrentCallingAddressPI()` Returns the PI that is associated with getCurrentCallingAddressPI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.

boolean	`getCurrentCallingDisplayNamePI()` Returns the PI that is associated with getCurrentCallingDisplayNamePI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.
boolean	`getLastRedirectingAddressPI()` Returns the PI that is associated with getLastRedirectingAddressPI. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.

The following interface on CiscoConnection retrieves the PI value for the address that is associated with the connection:

CiscoConnection

boolean	`getAddressPI()` Returns the PI that is associated with the address on which the connection gets created. If it returns true, the application displays the address name. If it returns false, the application must not display the address name.

No change exist in the message flow.

Privacy On Hold

This feature enhances the privacy of private held calls. When privacy is enabled, only the phone that placed a call on hold can retrieve that call, and the calling name and number are not displayed.

The feature provides the ability for a shared address to determine whether other shared addresses may barge into a call. When privacy is enabled, other shared address cannot barge into the call. Privacy is a terminals property. On IP phones, a Privacy feature button allows the users to enable and disable the privacy feature. Privacy can be dynamically enabled and disabled for the active calls on the terminal. When Privacy is on for a call, the TerminalConnection state available to other shared addresses is set to In Use. If Privacy status is changed during the CallProgress, CiscoTermConnPrivacyChangedEvent is delivered to the application.

In prior releases, if Privacy is enabled and the call is put on hold, all TerminalConnections were in TermConnHeld state and any other shared Address terminalConnection could unhold the call. In Cisco Unified Communications Manager 4.2, if the Enforce Privacy on Held Calls service parameter is enabled, and if Privacy is enabled for a call, putting the call on hold does not change the terminalConnections of other shared addresses and they remain in the In Use state.

Performance and Scalability

There is no performance impact with this feature because there is no additional traffic generated between Cisco Unified JTAPI, applications, and Cisco Unified Communications Manager.

Progress State Converted to Disconnect State

If an outbound call is initiated through the API to an unallocated directory number across the European PSTN, the application will perceive the ConnFailedEv event with the cause as CiscoCallEv.CAUSE_UNALLOCATEDNUMBER. For the US PSTN, the application may not see any event.

To make the behavior consistent across the European and American PSTNs and also to address backward compatibility issues, a new service parameter UseProgressAsDisconnectedDuringErrorEnabled was added to the jtapi.ini file starting with JTAPI Version 1.4(3.21), which, when enabled (1 = enable; 0 = disable; the default is disable), causes applications to perceive ConnFailedEv in both cases.

Q.Signaling (QSIG) Path Replacement

QSIG Path Replacement, a network feature, optimizes the real-time protocol (RTP) path when calls are transferred or forwarded to other PBXs that are connected through QSIG trunks. When path replacement is in progress, a small window of time exists when the feature requests from applications would be ignored and JTAPI would throw an exception to the application.

The Global Call ID or the call is changed when the RTP path is optimized with a direct path between the starting terminating PBXs. JTAPI provides new interfaces to monitor the call.

QoS Support

QoS support is enhanced in this release to enable QoS (DSCP marking) in both directions of the application <--> CTIManager connectivity. In previous releases it was enabled in only one direction: CTIManager --> application.

The DSCP (QoS) values for both directions of the link are set by the “DSCP IP CTIManager to Application” value in the CTIManager service parameters. The default value is CS3(precedence 3) DSCP (011000).

The “DSCP value for Audio calls” service parameter is the recommended QoS value for audio calls. Thisvalue is exposed to JTAPI applications.

You must perform one of the following setup procedures on the client machine for JTAPI QoS to work on Windows platforms.

Step 1 If you are running Windows 2000, follow the steps in QoS Setup on Windows 2000, on page 138.

Step 2 If you are running Windows XP or Windows Server 2003, follow the steps in QoS Setup on Windows XP Server 2003, on page 138.

What to do next

For more information on using the Registry Editor to set the Internet Protocol Type of Service bits, see the topic “Setsockopt is unable to mark the Internet Protocol type of service bits in Internet Protocol packet header” on the Microsoft technical support website.

These JTAPI interfaces support QoS:

Provider Interface

int	`getAppDSCPValue()` Returns the "DSCP IP for CTI applications" service parameter. This value specifies the DSCP value that JTAPI sets on its link to CTI. Applications can get this value by querying the provider object by using this API every time that they get a ProviderInServiceEvent.
private int	`precedenceValue = 0x00` Stores the DSCP value that CTI provides.

For details on these interfaces, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for QoS, see Message Sequence Charts, on page 755.

QoS Setup on Windows 2000

If you are running Windows 2000, follow these steps.

Step 1 Start the Registry Editor (Regedt32.exe).
Step 2 Go to key: HKEY_LOCAL_MACHINE on Local Machine\System\CurrentControlSet\Services\Tcpip\Parameters\
Step 3 On the Edit menu, click **Add Value**.
Step 4 In the **Value name** box, enter **DisableUserTOSSetting**.
Step 5 In the **Data Type** list, click **REG_DWORD** and then click **OK**.
Step 6 In the **Data** box, enter a value of **0** (zero) and then click **OK**.
Step 7 Quit Registry Editor and then restart the computer.

QoS Setup on Windows XP Server 2003

If you are running Windows XP or Windows Server 2003, follow these steps.

Step 1 Start Registry Editor (Regedt32.exe).
Step 2 Go to key: HKEY_LOCAL_MACHINE on Local Machine\System\CurrentControlSet\Services\Tcpip\Parameters\
Step 3 On the **Edit** menu, point to **New**, and then click **DWORD Value**.
Step 4 Enter **DisableUserTOSSetting** as the entry name, and then press **ENTER**.

When you add this entry, the value gets set to 0 (zero). Do not change the value.

Step 5 Quit Registry Editor and then restart the computer.

Quiet Clear

QuietClear occurs at the other end when two parties are on a call, and one address goes out of service because of a network outage, the Cisco Unified Communications Manager goes down, the application controlling CTIPort goes down, or CTIManager goes down. At this stage, the other end of the call can only drop the call or disconnect the connection. It cannot perform any other callControl operations.

For the party that went out of service, applications will perceive ConnDisconnectedEv and/or TermConnDroppedEv, and the other end of the call receives ConnFailedEv with CiscoCause of CiscoCallEv.CAUSE_TEMPORARYFAILURE.

If applications try to invoke the following features during QuietClear mode, PlatformException with error code of CiscoJtapiException.CTIERR_OPERATION_FAILER_QUIETCLEAR gets thrown:

- Consult transfer
- Consult conference
- Blind transfer
- Hold
- Unhold

Note Applications may only drop the call in this mode.

Receiving and Responding to Media Flow Events

Whenever a media stream must be created between two endpoints, Cisco Unified Communications Manager issues start transmission and start reception events to both endpoints. In JTAPI, the CiscoRTPOutputStartedEv and CiscoRTPInputStartedEv events represent the start transmission and start reception events. The CiscoRTPOutputStartedEv.getRTPOutputProperties() method returns a CiscoRTPOutputProperties object, from which the application can determine the destination address of its peer endpoint in the call, as well as the other RTP properties for the stream such as payload type and packet size. Similarly, the CiscoRTPInputStartedEv.getRTPInputProperties() method returns a CiscoRTPInputProperties object that informs the application of the RTP characteristics for the inbound stream.

At any time while media is flowing, the current CiscoRTPOutputProperties and CiscoRTPInputProperties also remain available from the CiscoMediaTerminal.getRTPOutputProperties() and CiscoMediaTerminal.getRTPInputProperties() methods as well. These methods throw an exception if the CiscoMediaTerminal is not currently supposed to transmit or receive media.

When Cisco Unified Communications Manager wants the application to stop sending or receiving media as the result of a call disconnecting or being put on hold, for example, it sends the CiscoRTPOutputStoppedEv and CiscoRTPInputStoppedEv events. These events mean that the current RTP media stream that exists between the two endpoints should be torn down.

Inbound Call Media Flow Event Diagram

The following table illustrates the dialogue between Cisco Unified Communications Manager and a JTAPI application when a call is presented to an application-controlled endpoint. The events in the left column represent JTAPI events that are sent to the CallObserver of the application, and the requests in the right column represent methods that the application invokes.

Table 4: Inbound Media Flow Event

JTAPI Event	Direction	Application Request
CallActiveEv ConnCreatedEv ConnProceedingEv CallCtlConnOfferingEv	Æ	
	¨	CallControlConnection.accept ()
CallCtlConnAlertingEv TermConnCreatedEv TermConnRingingEv	Æ	
	¨	TerminalConnection.answer ()
ConnConnectedEv CallCtlConnEstablishedEv TermConnTalkingEv CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	Æ	
	¨	CallControlConnection.disconnect ()
CiscoRTPOutputStoppedEv CiscoRTPInputStoppedEv TermConnDroppedEv CallCtlConnDisconnectedEv	Æ	

Note The table above shows JTAPI events for the local connection: that is, for the application endpoint. TheactualJTAPI meta event stream contains events that describe the state of the calling party.

Cisco Unified Communications Solutions RTP Implementation

The Cisco Unified Communications Solutions architecture puts a premium on performance, and thus Cisco Unified Communications Solutions phones and gateways do not implement some of the features of RTP and its often-associated real-time control protocol (RTCP). To ensure its compatibility, applications must consider the following points:

- Because RTCP is not supported. Cisco Unified Communications Solutions endpoints will not send RTCP messages, and they will ignore any such messages that are sent to them.
- Cisco Unified Communications Solutions endpoints do not currently make use of the synchronization source (SSRC) field in the RTP header. Applications must not multiplex RTP streams by using the SSRC field, or phones and gateways may not correctly decode and present the media.

Recording

Introduction

New regulations require organizations to archive contact interactions to meet compliance directives and Contact Centers need to guarantee the quality of service their Agents provide. Cisco's Recording feature enables organizations to archive the conversation of two or more parties for review, analysis, and/or legal compliance.

The recording feature lets applications record conversations on any observed address. Three recording configurations are available:

- No recording
- Automatic recording:

  The system initiates a recording session and streams media to the configured recording device whenever a call goes to a connected state.

- Application-controlled recording:

  If application-controlled recording is configured on an address, the application can start and stop recording. The call must exist in the connected state before the application can start recording.

The ability to record calls was introduced in Unified Communications Manager Release 6.0 with phone-based built-in bridge (BIB) recording. Cisco IP Phones were instructed to send copies of conversations to supervisors and recorders. In Release 8.0, Encrypted media (sRTP) support was added and was expanded to have the information sent to recorders (meta-data) in Release 8.6(2). With Release 9.0 selective user controlled recording was added to the feature

In Unified Communications Manager Release 10.0(1), the recording feature is enhanced so that Dynamic combinations of Cisco Gateways and IP Phone are instructed to send copies of conversations to recorders based on call flows, participants, and media requirements. Also, recording Serviceability counters and alarms have been added to help compliance officers ensure calls are recorded by monitoring the real-time status and historical performance of the feature.

For internal calls within a cluster between end users, the media-forking device is the end user device involved in the call that triggers the recording session. For an external call from a recording gateway, both the end user device and the gateway involved in the call can be used as the media-forking device. Unified Communications

Manager enables the administrator to select one over the other as the preferred recording media source: "Phone preferred" or "Gateway preferred", using the device's line configuration.

If the phone preferred recording media source is selected, the phone that triggers the recording is used to fork the media for the recording session, provided that the phone is capable of media forking and phone's BIB is enabled. If the phone is not capable of media forking or the phone's BIB is not enabled, and the gateway involved in the call has the media forking capability and is enabled for recording, then the gateway is used to fork the media for the recording session. If the gateway preferred recording media source is selected, the gateway that is already in the media path will be used to fork the media for the recording session, provided that the gateway is capable of media forking and is enabled for recording. If gateway is not capable of media forking or is not enabled for recording, and the phone triggering the recording is capable of media forking and the phone's BIB is enabled, then the phone is used to fork the media for the recording session. If none of the recording resources is available, this recording request fails. Similar to the phone-based recording, gateway-based recording is also triggered from the end-user's device or CTI/JTAPI application.

Note Virtual devices without BIB such as CTI Port, Route Point, and CTI Remote Device can only be set to Gateway-Preferred.

When a gateway is registered with the same cluster as the device that initiates a recording, it is called a "Single Cluster" gateway recording. When a gateway is registered with a cluster that is different than where the recording request is initiated, it is called an "Inter-Cluster" gateway recording. The cluster where the recording initiates is a **recording triggering cluster** (Trigger) and the cluster where the recording gateway registers to is a **recording anchoring cluster** (Anchor).

Note The inter-cluster recording is only supported by SIP trunks.

When there is any mid-call feature involved with the call being recorded, the recording resource may change due to the feature interactions. In previous Unified Communications Manager releases, the recording sessions started by a near-end party continues when the far-end party can hold or transfer the call while the near-end party remains connected. The only time the recording session restarts is when the near-end party holds and resumes the call. However, for Gateway-based Recording, Unified Communications Manager no longer maintains this behavior. Instead of continuing the recording session when the connected party of the near-end changes, the recording session is re-started by the near-end party. In JTAPI, this "Recording-Re-trigger" behavior results in extra **CiscoTermConnRecordingEndEv** and **CiscoTermConnRecordingStartEv** events sent to applications. With the new behavior, each recording session is a complete section of the conversion between two unique parties. A near-end connected party change can be caused by mid-call features such as call transfer, call redirect, conference, shared line hold/resume, etc. Therefore, from JTAPI application perspective, there can be multiple RecordingStart/Stop events within a single call. This applies to both Gateway-based recording and Phone/BIB-based recording.

In Cisco Unified Communications Manager Release 10.0(1), CTI is introducing support for Gateway Recording (in addition to existing phone-based BIB Device Recording). CTI applications, using Cisco JTAPI, is able to differentiate a recording call's recording type and media forking device/cluster info from existing JTAPI interface and event; and a new JTAPI event is also introduced to identify recording failure as described below. With this new Gateway Recording feature, either the gateway or the built-in bridge (BIB) can be used as the recording resource based on the end user's preference and the availability.

The following interfaces extend TermConnEv and are delivered to callobserver. For shared lines, the system delivers these events to call observers on the address or terminal of the talking terminal connections.

Applications receive no events if they have only the terminal whose connection is in the INUSE or BRIDGED state.

CiscoTermConnRecordingStartEv

`CiscoTermConnRecordingStartEv`

Indicates the start of recording and is delivered to the call observer of the recording initiator. Auto recording configuration or an application request can trigger recording.

CiscoTermConnRecordingEndEv

`CiscoTermConnRecordingEndEv`

Indicates the end of recording and is delivered to the recording initiator.

CiscoTermConnRecordingFailedEv

`CiscoTermConnFailedEv`

This interface is added for Cisco Unified Communications Manager Release 10.0(1) and indicates when a call recording failed.

Exposing Recording Media Forking Info on CiscoRecorderInfo

Cisco JTAPI provides new APIs for Release 10.0(1): getMediaForkingDeviceType(), getMediaForkingDeviceName(), getProtocolReferenceGUID(), and getMediaForkingClusterID() to expose various call recording media forking information of a recording call to JTAPI application. These capabilities are exposed on the existing interface of CiscoRecorderInfo, where applications can extract from **CiscoTerminalConnection** and **CiscoTermConnRecordingTargetInfoEv**.

Exposing Recording Media Forking Device Type on CiscoCall

With Release 10.0(1), Cisco JTAPI introduces three new forking device types:

- CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_NONE
- CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE
- CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

Exposing Cluster ID on CiscoProvider

Cisco JTAPI provides API of CiscoProvider.getClusterID(), which returns the clusterID enterprise parameter configured for the cluster. (Note that the cluster ID is an Enterprise parameter configurable from CUCM admin page, and when this parameter is changed by administrator, the CTIManager service and CallManager service would need to be restarted for it to take effect).

Secured Recording

With this enhancement a recording device can record a secure call if its device security capability is same as or more than that of the agent. A recording request fails if the recording is attempted for an authenticated device, or if the security capability of the recorder is non-secured and that of the agent is encrypted.

Backward Compatibility

This feature is not backward compatible and existing applications can be affected with the introduction of this new feature. That is, when mid-call feature(s) is involved, there can be recording retrigger(s) with multiple recording sessions within a single call, applications need to coordinate these recording sessions accordingly. This new change of behavior applies to both Gateway-based recording as well as Phone/BIB-based recording.

For detailed information about these interface changes, see the following topics:

- CiscoJtapiException, on page 410
- Related Documentation, on page 283
- CiscoCall, on page 326
- CiscoProvider, on page 486
- CiscoProviderCapabilities, on page 498
- CiscoProviderCapabilityChangedEv, on page 500
- CiscoProviderObserver, on page 502
- CiscoRecorderInfo, on page 505
- CiscoTerminalConnection, on page 630
- CiscoTermConnRecordingTargetInfoEv, on page 588

Redirect

JTAPI 1.2 specifies that one of the preconditions of the CallControlConnection.redirect() method specifies for the state of the connection to be in either the CallControlConnection.OFFERING or the CallControlConnection.ALERTING state. Cisco Unified JTAPI also allows a connection in the CallControlConnection.ESTABLISHED state to get redirected.

The redirect() method includes the following overloaded form in the CiscoConnection interface. It allows applications to specify the behavior that is desired when a failure occurs while a call is redirected and specifying the calling search space, or resetting the original called field.

Applications choose the desired behavior, by passing one of the following INT parameters in the overloaded redirect method from the CiscoConnection interface:

- Redirect drop on failure—When a call is directed to a busy or an invalid destination, Cisco Unified Communications Manager can either drop the call if the redirect fails or leave the call at the redirect controller. The JTAPI application can then take corrective action, such as redirecting the call to another destination. The option for the redirect mode parameter follows:
  - CiscoConnection.REDIRECT_DROP_ON_FAILURE
  - CiscoConnection.REDIRECT_NORMAL
- Calling Address search space—Redirect uses the calling search space parameter to indicate which callingSearchSpace is used. Applications can either use the calling party search space or the redirect controller search space. The parameter options for this scenario follow:
  - CiscoConnection.CALLINGADDRESS_SEARCH_SPACE
  - CiscoConnection.ADDRESS_SEARCH_SPACE
- Resetting original called—The called address option parameter gets used to reset the original called fields. The options for this scenario follow:
  - CiscoConnection.CALLED_ADDRESS_UNCHANGED

- CiscoConnection.CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION. This option affects the fields when the call arrives at the redirect destination.

For more information, refer to the com.cisco.jtapi.extensions.CiscoConnection documentation.

For the scenario where A Calls B, B redirects to C, and C (redirect destination) does not represent a provider observed address, JTAPI would provide CallCtlConnAlertingEv for C with cause code Ev.CAUSE_NORMAL. Prior to release 5.0, the cause code specified Ev.CAUSE_REDIRECT for the same scenario.

This change kept the behavior consistent for scenarios where C observed or did not observe the provider.

Note When C is observed, for the same scenario, CallCtlConnAlertingEv at C is provided with CAUSE_NORMAL from releases prior to 5.0, and that behavior continues without change.

Redirect Set Original Called ID

Cisco Unified JTAPI applications can specify the preferred original called party DN in the redirect request. The Redirect Set Original Called ID feature lets applications redirect a call on a connection to another destination while letting the applications set the OriginalCalledID to any value. This enables applications to transfer the call directly to the voice mail of another. For example, if A calls B and B wants to transfer the call to CVoice Mail, applications can specify in the enhanced redirect request C as the preferred original called party and destination party as CVoice Mail profile. With this request, calls appear in C Voice Mail profile with the Cisco Unified Communications Manager originalCalledParty field as C. Typical voice mail applications look for originalCalledParty information to identify a user voice mailbox.

Any application that redirects a call to a party by modifying the original called party can take advantage of this feature.

Note This feature also changes the lastRedirectedAddress to the preferredOriginalCalledParty that gets specified in the redirect request.

The following callControlConnection interface applies for Redirect Set Original Called ID:

```
Interface CiscoConnection Extends callControlConnection With Additional Cisco Unified
Communications Manager-Specific Capabilities
```

javax.telephony.Connection	**redirect (java.lang.String destinationAddress, intmode, int callingSearchSpace, java.lang.String preferredOriginalCalledParty)** This method overloads the CallControlConnection.redirect() method.

For details on the interface, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for Redirect Set Original Called ID, see Message Sequence Charts, on page 755

Redirect to Device

With Release 11.5(1), the Redirect feature of Cisco Unified JTAPI is enhanced to allow you to redirect calls to a specific device via the deviceName parameter. Even in shared line situations, the redirected call goes to the target device only and not to other devices that share the phone line.

To support this feature, the following methods have been enhanced with a new deviceName field:

- CiscoConnection.redirect now includes a deviceName field, which allows you to target the redirect to a specific device. If another device shares the same phone line, that device goes into remote-in-use state. Cisco JTAPI delivers a TermConnPassiveEv and CallCtlTermConnInUseEv to the shared line devices.
- CiscoRouteSession.selectRoute also includes a deviceName field allowing the selectRoute() method to take an array of destination device names. The order of device names corresponds to the order of route selected. Once the route is selected, Cisco JTAPI attempts to redirect the call to the destination device.

Table 5: Method Structure

Interface	Method
CiscoConnection	redirect(String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, String preferredOriginalcalledParty, String facCode, String cmcCode, int featurePriority, byte[] applicationXMLData, String deviceName)
CiscoRouteSession	selectRoute(String[] routeSelected, int[] callingSearchSpace, String[] modifyingCallingNumber,String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption, String[] facCode, String[] cmcCode,int[] featurePriority, byte[][] applicationXMLData, String[] deviceName)

Restrictions

The following restrictions apply:

- If an invalid deviceName is passed to the redirect method, the REDIRECT_CALL_INVALID_DEVICE_NAME error gets thrown.
- The deviceName can be used to redirect calls within the cluster only. If the application attempts to redirect a call across clusters with the deviceName completed, the REDIRECT_CALL_INVALID_DEVICE_NAME gets thrown. To redirect calls across clusters, the deviceName must be null, or the application must use other redirect methods.
- The deviceName must be associated to the directory number that the application passes to the redirect method and not any other directory number. Otherwise, the REDIRECT_CALL_INVALID_DEVICE_NAME error gets thrown

Backward Compatibility

There is no impact on backward compatibility as the above methods are overloaded.

Message Sequence Charts

Redirect to a Device, on page 1443

Redundancy

Configuration requires that devices are configured into device pools and are assigned static Cisco Unified Communications Manager groups. Devices register with a particular Cisco Unified Communications Manager server that handles call control signaling. When a server fails, the devices failover to the backup server in the group. When the primary server comes back online, it waits until no active calls exist on the device, then re-homes to the primary Cisco Unified Communications Manager server. Cisco Unified JTAPI informs the applications of this transition by sending a temporary out-of-service message while registering to the backup server.

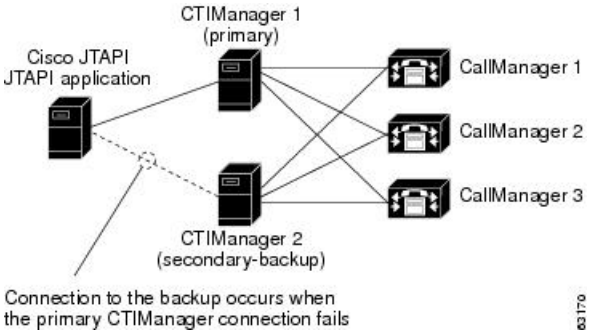
Redundancy in CTI Managers

Cisco Unified JTAPI also offers transparent applications for redundancy via the CTI Manager. When the primary CTIManager fails, Cisco Unified JTAPI automatically connects to the backup CTI Manager and communicates the reconnection to applications. Instead of connecting to a single Cisco Unified Communications Manager server, applications now connect to a set of CTIManagers. The applications supply the CTIManager server names when they invoke JTAPI.

Cisco Unified JTAPI and the CTIManager maintain bidirectional heartbeat signals to detect a loss of connectivity between them. The CTIManager detects when an application no longer runs and cleans up its allocated resources. The following figure illustrates the“Logical Representation of JTAPI, CTIManager and Cisco Unified Communications Manager in a cluster”

Note After Cisco Unified JTAPI successfully connects to the primary CTIManager, it alternately will attempt to reconnect to the primary or backup CTIManager if the JTAPI connection to the CTIManager fails.

Figure 12: Logical Representation of JTAPI, CTIManager and Cisco Unified Communications Manager in a Cluster



Invoking CTIManager Redundancy

When getProvider() method on the CiscoJtapiPeer is called during the application startup, Cisco Unified JTAPI attempts a connection to the first CTIManager in the list and tries a connection to the next CTIManager

if connection attempt fails with the first. If all the CTIManagers in the list are not available or if connection is refused by all CTIManagers, an exception gets sent to the application, and no further reconnection attempts occur. After the first successful connection, Cisco Unified JTAPI alternatively attempts to connect to the backup or primary CTIManager when a failure to CTIManager or connection to CTIManager is detected.

The list of redundant CTIManagers designates a comma-separated list that is passed into the CiscoJtapiPeer.getProvider(String providerString) method as a String. The usage for the providerString follows:

- providerString = CTIManager;login = XXX;passwd = YYY;appinfo = ZZZ (Non-redundant feature)
- providerString = CTIManager1, CTIManager2;login = XXX, passwd = YYY;appinfo = ZZZ (Redundant feature)

Note Because the appinfo parameter is optional, the application provides no specific appinfo parameter. Cisco Unified JTAPI generates one from a JTAPI instance ID and the local host name.

Additionally, the jtapi.ini file may define different CTIManager lists to support the CiscoJtapiPeer.getServices() method. Cisco Unified JTAPI accepts the following definition:

CtiManagers = <CTIManager1>, <CTIManager2>;<CTIManager3>

where

<CTIManager1>, <CTIManager2> specifies a redundant group.

<CTIManager3> specifies a nonredundant group.

From Unified CM Release 14SU3 onwards, support has been added to allow an application to specify a CTIManager as having **least priority**. Prior to this, all CTIManagers in a redundancy group have equal weightage. JTAPI would attempt to failover to the next availabe CTIManager in the group, if connection is lost or not established to the current server.

This feature support is added to aid Dedicated Instance (DI) deployments that are cloud managed. It has been extended as a general usage API for all applications.

An application needs to invoke setLeastPriorityCtiServer exposed on the <CiscoProvider>.

Once a CTIManager is marked as least priority, JTAPI includes the configured CTIManager internally into the redundancy group. JTAPI attempts a connection to this CTIManager only if no other CTIManager is available.

Once connected to the least priority CTIManager, JTAPI would deliver a CiscoProvConnToLeastPriorCtiServerEv on provider to indicate it is now connected to least priority CTIMaanger.

JTAPI internally monitors availability or reachability to the other servers in the group.

Once one of the servers are available, JTAPI would deliver a CiscoProvPrimNwReachableEv on the Provider observer to indicate one of the other servers are reachable now.

JTAPI would later attempt a failover based on the configured fallback initiation time as specified by application via the API.

If no time is specified, it would default to 10 min, post which JTAPI would forcefully failover application to the CTIManager which is available now.

On successful fallback, JTAPI would deliver a CiscoProvFallbackToPrimNwCompltdEv on Provider to indicate it is no longer connected to the least priority CTIManager server.

CTIManager Failure

When Cisco Unified JTAPI detects a loss of connection to a CTIManager, the application receives notification of this loss in service. The following events get sent to the application on the appropriate Observers:

- A CallObservationEndedEv event gets sent to all call observers on an address, and calls in progress end. The calls get physically connected, but the application observation of the call ends because Cisco Unified JTAPI cannot send call state changes.
- A CiscoAddrOutOfServiceEv event gets sent to all addresses on a terminal and a CiscoTermOutOfServiceEv event gets sent to the terminal.
- This process repeats for all terminals in the provider user-controlled list. (A CiscoAddrOutOfServiceEv event gets sent only to the addresses that have an active AddressObserver, and a CiscoTermOutOfServiceEv event gets sent only to terminals with an active TerminalObserver.)
- The provider gets set in the out-of-service state, and the ProvOutOfServiceEv event gets delivered on any ProviderObserver callbacks present on the provider.

Cisco Unified JTAPI attempts a connection to the next CTIManager in the list, and the ProvInServiceEv gets sent to the ProviderObserver. The devices that previously registered under the application control get reinstated in the new CTIManager After the device is reinstated, CiscoAddrInServiceEv and CiscoTermInServiceEv events get sent to the application via the respective observers. All previously added observers are maintained. If any calls exist on the devices, a snapshot of the call gets sent to the respective call observers.

CTI ports that were previously registered are reregistered with the same media parameters. RouteAddress callbacks are maintained as before, and these calls get recovered on the new CTIManager. No call snapshot, however, gets delivered to the RouteAddresses.

If a least priority CTIManager was set, and an application fails over to it, JTAPI delivers a CiscoProvConnToLeastPriorCtiServerEv on Provider.

Heartbeats

Cisco Unified JTAPI and the CTIManager maintain heartbeat signals to discover a failure in either the CTIManager or JTAPI. The CTIManager server controls the heartbeat parameters in the bidirectional heartbeat. Applications can request a desired server heartbeat interval when they are initializing Cisco Unified JTAPI, but the CTIManager can override it.

Applications specify the desired heartbeat parameter by using DesiredServerHeartbeatInterval in the jtapi.ini setting.

Cisco Unified JTAPI specifies the desired heartbeat interval for the client during initialization. The CTIManager specifies the client side heartbeat interval to Cisco Unified JTAPI and specifies the interval at which the server (CTIManager) will send heartbeats. A failure to receive heartbeat message for twice the server-specified interval results in a client-initiated teardown of the connection. To minimize heartbeat traffic, any messages from the client to the server or events from the server to the client substitute for a heartbeat.

Ringback on SIP 183 for Transferred Calls

In Release 11.0(1), Cisco JTAPI has been updated with how it responds to SIP 183 messages when a call is transferred over a gateway or trunk. When an established call gets transferred over a trunk and a SIP 183 is received, Cisco JTAPI moves the call to CallControlConnection.NETWORK_ALERTING state. When the call is answered, Cisco JTAPI moves the call state to CallControlConnection.ESTABLISHED.

A new Cisco CallManager service parameter, **CTI Report Ringback on SIP 183 with SDP**, has been added to configure this feature. When this sevice parameter is set to **True**, the above behavior applies. This is the default setting.

If an application needs to use the legacy behavior, you can set the service parameter to **False**. Under this setting, if the call call is transferred over a gateway or trunk, CTI will use the CallControlConnection.NETWORK_REACHED state to report that the other network has been reached, but CTI will not report back that a connection has been established.

Routing

Routing in JTAPI requires the configuration of a CTI Route Point on the Cisco Unified Communications Manager. Multiple calls can be queued to this Route Point, but only a single line can be configured on a CTI Route Point device.

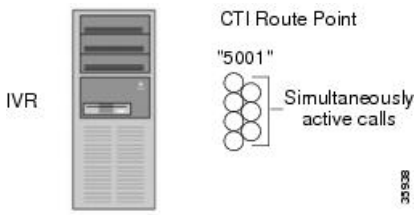
JTAPI implementation of adjunct Routing, as described in the call center package, includes the following actions:

- Registering route callbacks on Route Addresses
- Creating appropriate handlers in response to the various routing events (routeSelect, routeEnd)

Note CTI Route Points represent devices that can process any number of incoming calls simultaneously on the same line. You can route calls by using the methods in the javax.telephony.callcenter package, or you can accept, redirect, or disconnect calls by using the methods in the javax.telephony.callcontrol package. You can configure each CTI Route Point with a maximum of 34 lines. To support more than 34 lines, provision additional route points. For details on how to configure and administer the CTI Route Point, refer to the Cisco Unified Communications Manager Administration Guide.

The following figure shows the CTI Route Point configuration.

Figure 13: CTI Route Points



Cisco Route Session Implementation

When a call comes in to the RouteAddress, the implementation starts a Route Session thread and sends the application a RouteEvent. This thread in turn starts a timer thread to time the application response to a RouteEvent with either a routeSelect() or an endRoute(). If the application responds with a routeSelect (String[] selectedRoutes), JTAPI verifies that all preconditions are satisfied and then attempts to route the call to the first destination that is specified in the array. If the destination is a valid and available number, the call gets routed, and the application gets a RouteUsedEvent followed by a RouteEndEvent. Otherwise, if an error occurs in routing (which may be caused by an invalid/busy/unavailable destination), the application gets a ReRouteEvent. JTAPI starts the Timer Thread again before it sends the re-Route Event. Because Cisco Unified Communications Manager does not support re-Routing, if the routing was unsuccessful, either the caller will receive a busy tone, or the call will get dropped. The application can clean up all failure instances and/or send JTAPI an endRoute to clean up the RouteSession. If the application does not respond with an endRoute(), the JTAPI timer once again expires, and JTAPI cleans up the Route Session by sending the application a RouteEndEvent().

If the routing timer expires before the application returns with a selectRoute() or an endRoute() method, the Cisco Unified Communications Manager applies same treatment as when a call is made to an unregistered phone (that is, play fast busy). If ForwardNoAnswer is configured on the Route Point, the call immediately forwards to that number when the timer expires.

If the application cannot respond with a valid address to which to route the call, the application may choose to call endRoute with an error. The JTAPI specification defines three errors in the RouteSession interface: ERROR_RESOURCE_BUSY, ERROR_RESOURCE_OUT_OF_SERVICE, and ERROR_UNKNOWN. If an endRoute is invoked on the RouteSession, the implementation currently accepts() the call at the RouteAddress, so the caller may begin to receive ringback. If forwarding is configured for the Route Point, the call gets forwarded when the Forwarding Timer expires.

Select Route Timer

Configure this timer via the JTAPI.ini configuration file that has a key called RouteSelectTimeout = 5000. Use milliseconds as the unit. The default value for this timer specifies 5 seconds; however, depending on the needs of the application, you can extend or decrease this timer to improve Route Session cleanup efficiency. Ensure that this timer is not unreasonably large. Each Route Session as a thread represents a call to the Route Point, and these Route Sessions should be cleaned up. Should an application expect significant delays between receiving the Route Event and responding with a routeSelect/endRoute event, the application would want to appropriately extend this timer.

Forwarding Timer

You can configure the timer for Forward on No Answer that is currently systemwide (that is, it applies to all devices on Cisco Unified Communications Manager) via the Cisco Unified Communications Manager Service Parameters configuration. The default value for this timer specifies 12 seconds. In future releases, a separate timer for CTI Route Points might get included, so forwarding for the route point takes effect immediately after JTAPI accepts the call (when the application calls an endRoute or if the routing timer expires).

Route Session Extension

CiscoRouteSession acts as a Cisco Extension to the JTAPI specification. Most importantly, this extension exposes the underlying Call object to the Applications. CiscoRouteSession.getCall() returns CiscoCall, and

this call exposes other Call Model Objects such as the associated Addresses, Connections, and so on. The extension also defines additional errors for the application.

Caller Options Summary

In the absence of a callback, or if RouteSession.routeSelect() or endRoute() has not responded to a routeEvent, the caller receives nothing until

- The application can disconnect() or reject() the connection on the Route Point, and, thereby, the caller receives a busy tone.
- The application can accept the call, and the Forward No Answer, if configured, kicks in.
- The application can drop the call. The caller holds the receiver but does not know what happened.

With a callback, if the application chooses to call an endRoute(), after endRoute() returns, the caller receives a ringback until

- The client calls a disconnect() that would drop the call.
- The client redirects() the call.
- The forward on no answer timer that is configured via the scm.ini will kick in and forward the call unless the preceding two options have already kicked in.
- If no forwarding is configured for the Route Point, the caller continues to receives a ringback unless the first two options kick in.

Fault Tolerance When Using Route Points

One way for an application that uses route points to deal with fault tolerance requires connecting two JTAPI applications to two different Cisco Unified Communications Managers, each registering a different RouteAddress. For example, Application1 manages RouteAddress1 by using Communications Manager1. Application2 manages RouteAddress2 by using Communications Manager2. In Cisco Unified Communications Manager Administration, ensure the ForwardNoAnswer configuration for these CTI Route Points is administered, so they point to each other. In this example, RouteAddress1 would have FNA = RouteAddess2, and RouteAddress2 would have FNA = RouteAddress1. If Communications Manager1 goes down, calls forward to RouteAddress2, so Application2 takes over. Furthermore, both applications could be configured to reconnect to the proper Cisco Unified Communications Manager server when they receive a ProviderShutdown event.

Secure Conferencing

This feature informs applications whether a call is secure, allowing for secure conference calls. When the overall security status of the call changes, secure conferencing provides applications with a notification in the form of an event on the call. Applications receive the overall call security status of the call in the CiscoCallSecurityStatusChangedEv when the overall call security status changes. When a terminal goes to the talking state, JTAPI provides the call security status information to the applications. Applications can query the security status of the call by using a new interface on CiscoCall. The system makes the security status information available to applications when the applications start monitoring an existing call.

In shared address scenarios, the system also reports CiscoCallSecurityStatusChangedEv to the RIU parties. The OverallCallSecurityStatus matches the status reported on the active terminals. For example, in a three-party conference with A (Encrypted), B (Encrypted), C (Authenticated), and C' (Authenticated), the system reports CiscoCallSecurityStatusChangedEv with OverallCallSecurityStatus = Authenticated to C and C'. The system delivers this event on a per-call basis.

SRTP key information will continue to be sent for encrypted parties whether or not the OverallCallSecurityStatus is Encrypted. For example, in a three-party conference with A (Encrypted), B (Encrypted), and C (non-secure), the OverallCallSecurityStatus of the conference call is NotAuthenticated. However, the media that connects A, B, and the conference bridge continues to be encrypted because they are encrypted parties. Thus, A and B receive SRTP keys despite the OverallCallSecurityStatus.

Backward Compatibility

This feature is backward compatible. The new parameter, EnableSecurityStatusChangedEv, in the jtapi.ini file controls the new event CiscoCallSecurityStatusChangedEv that the secure conferencing feature generates. Applications can turn on this parameter by adding the line "EnableSecurityStatusChangedEv = 1" to the jtapi.ini file to receive this new event. By default, this parameter does not appear in the jtapi.ini file, so event notification is disabled. The setCallSecurityStatusChangedEv() interface on com.cisco.jtapi.extensions.CiscoJtapiProperties lets applications set this ini parameter programmatically.

For additional information, see CiscoCallSecurityStatusChangedEv, on page 362.

Secure Real-Time Protocol Key Material

This feature provides the mechanism that is needed to deliver Secure Real-Time Protocol (SRTP) key material of an encrypted media session between authenticated end points within Cisco Unified Communications Manager based Enterprise systems. To receive this key material, the administrator must configure the TLS Enabled and SRTP Enabled flags in the Cisco Unified Communications Manager Administrator windows and a TLS link must be established between JTAPI and the CTIManager.

Key materials get exposed in CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv. To get these events, applications must enable rtpKeyEvenabled in CiscoTermEvFilter. By default, filters are disabled to maintain backward compatibility. If filters are enabled, application always get CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv. A security indicator in these events indicates whether the media is encrypted and whether keys are available.

CiscoRTPInputKeyEv contains key material of the input stream and CiscoRTPOutputKeyEv contains key material of the output stream. Applications can use this key material to decrypt the packets and start monitoring or recording the media. Applications must not store this key material in a way that leaves the material vulnerable to tampering, and applications must zero out or clear the entry for these keys when they go out of scope.

This key material contains

- Key Length
- Master Key
- Salt Length
- Master Salt
- AlgorithmID
- isMKIPresent

- Key Derivation Rate

This enhancement also supports a secure media termination for CTIPorts and RoutePoints. To do this, the application passes in supported encrypted algorithms in CTIPort and route point register requests. The application gets an error if no TLS link and no SRTP Enabled flags exist. Whether media are encrypted or not depends on whether the other end is interested in secure media and whether the algorithm is negotiated successfully.

For mid-call monitoring, if the application comes up after a call is established between two end points, the application must query Terminal.createSnapshot() and snapshot event CiscoTermSnapshotEv. CiscoTermSnapshotCompletedEv gets sent, which indicates whether the current media between end points is secure or not. Applications can query CiscoMediaCallSecurityIndicator to get a security indicator for a call; however, this does not contain any key material in the event. If no calls exist on any of the lines on the terminal, applications only get CiscoTermSnapshotCompletedEv. To maintain backward compatibility, these events get generated only when an application enables the snapShotRTPEnabled filter in CiscoTermEvFilter.

CiscoRTPHandle gets added in all RTP events so that applications can correlate RTP events related to a single call. For backward compatibility, no new events are generated when there is no secure media.

For more information on SRTP, see the Secure RTP Library API Documentation by David McGrew on SourceForge.net.

The following sections describe the interface changes for SRTP key material.

Public Interface CiscoMediaEncryptionKeyInfo

int	`getAlgorithmID()` This method returns the media encryption algorithm for the current stream.
int	`getIsMKIPresent()` An MKI indicator that indicates whether MKI is present. Key management defines, signals, and uses the MKI.
int	`getKeyLength ()` This method returns the master key length.
byte[]	`getKey()` This method returns the master key for the stream.
int	`getSaltLength ()` This method returns the salt length.
byte[]	`getSalt()` This method returns the salt key for the stream.
int	`keyDerivationRate()` Indicates the SRTP key derivation rate for this session.

CiscoMediaSecurityIndicator

static int	`MEDIA_ENCRYPTED_KEYS_AVAILABLE` Indicates that media terminated is secured and keys are available.
static int	`MEDIA_ENCRYPTED_KEYS_UNAVAILABLE` Indicates that media is terminated in secured mode, but keys are not available because SRTP is not enabled in Cisco Unified Communications Manager Administration User windows. This could be because either no TLS exists or no IPSec is configured for this application.
static int	`MEDIA_ENCRYPTED_USER_NOT_AUTHORIZED` Indicates that media is terminated in secured mode, but keys are not available because user is not authorized to get the keys.
static int	`MEDIA_NOT_ENCRYPTED` Indicates that media is not encrypted for this call.

CiscoRTPInputKeyEv

CiscoMedia EncryptionKeyInfo	`getCiscoMediaEncryptionKeyInfo ()` Returns CiscoMediaEncryptionKeyInfo only if the provider is opened with TLS link and if SRTP enabled option is set for the application in Cisco Unified Communications Manager User Administration; otherwise, it returns null.

int	`getCiscoMediaSecurityIndicator()`
	Returns media security indicator, which is one of the following constants from the CiscoMediaSecurityIndicator:
	`MEDIA_ENCRYPTED_KEYS_AVAILABLE`
	`MEDIA_ENCRYPT_USER_NOT_AUTHORIZED`
	`MEDIA_ENCRYPTED_KEYS_UNAVAILABLE`
	`MEDIA_NOT_ENCRYPTED`

CiscoCallID	`getCallID ()` Returns CiscoCallID object if CiscoCall is present when this event is sent. If no CiscoCall is present, this method returns null.
CiscoRTPHandle	`getCiscoRTPHandle ()` Returns CiscoRTPHandle object. Applications can get a call reference by usingCiscoProvider.getCall(CiscoRTPHandle). If no call observer exists, or if there was no call observer when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoRTPOutputKeyEv

CiscoMedia EncryptionKeyInfo	`getCiscoMediaEncryptionKeyInfo ()` Returns CiscoMediaEncryptionKeyInfo only if the provider is opened with TLS link and if the SRTP enabled option is set for the applicationin Cisco Unified Communications Manager User Administration. Otherwise, it returns null.

int	`getCiscoMediaSecurityIndicator()`
	Returns media security indicator, which is one of the following constantsfrom CiscoMediaSecurityIndicator:
	`MEDIA_ENCRYPTED_KEYS_AVAILABLE`
	`MEDIA_ENCRYPT_USER_NOT_AUTHORIZED`
	`MEDIA_ENCRYPTED_KEYS_UNAVAILABLE`
	`MEDIA_NOT_ENCRYPTED`

CiscoCallID	`getCallID ()` Returns CiscoCallID object if CiscoCall is present when this event is sent. If no CiscoCall is present, this method returns null.
CiscoRTPHandle	`getCiscoRTPHandle ()` Returns CiscoRTPHandle object. Applications can get a call reference by usingCiscoProvider.getCall(CiscoRTPHandle). If no call observer exists, or if there was no call observer when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoTermSnapshotEv

CiscoMediaCall MediaSecurity Indicator[]	`getMediaCallSecurityIndicator ()` Returns media security status for each active call on this device.

CiscoTermSnapshotCompletedEv

This event has no methods.

CiscoMediaCallSecurityIndicator

int	`getCiscoMediaSecurityIndicator()`
	Returns media security indicator, one of the following constants from CiscoMediaSecurityIndicator:
	`MEDIA_ENCRYPTED_KEYS_AVAILABLE`
	`MEDIA_ENCRYPT_USER_NOT_AUTHORIZED`
	`MEDIA_ENCRYPTED_KEYS_UNAVAILABLE`
	`MEDIA_NOT_ENCRYPTED`

CiscoCallID	`getCallID ()` Returns a CiscoCallID object if a CiscoCall is present when this event is sent. If no CiscoCall is present, this method returns null.
CiscoRTPHandle	`getCiscoRTPHandle ()` Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall(CiscoRTPHandle). If no callobserver exists or if there was no callobserver when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoRTPInputStartedEv

CiscoRTPHandle	`getCiscoRTPHandle ()` Returns a CiscoRTPHandle object. Applications can get a call reference by usingCiscoProvider.getCall(CiscoRTPHandle). If no call observer exists, or if there was no call observer when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoRTPInputStoppedEv

CiscoRTPHandle	`getCiscoRTPHandle ()` Returns a CiscoRTPHandle object. Applications can get call reference by usingCiscoProvider.getCall(CiscoRTPHandle). If no call observer exists, or if there was no call observer when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoRTPOutputStartedEv

CiscoRTPHandle	`getCiscoRTPHandle ()` Returns a CiscoRTPHandle object. Applications can get a call reference by usingCiscoProvider.getCall(CiscoRTPHandle). If no call observer exists, or if there was no call observer when this event is delivered, CiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoRTPOutputStoppedEv

CiscoRTPHandle	`getCiscoRTPHandle ()` Returns CiscoRTPHandle object. Applications can get call reference usingCiscoProvider.getCall(CiscoRTPHandle). If there is no call observer, or if there was no call observer when this event is delivered, thenCiscoProvider.getCall(CiscoRTPHandle) may return null.

CiscoTermEvFilter

boolean	`getSnapshotEnabled ()` Returns the enable/status of CiscoTermSnapshotEv and CiscoTermSnapshotCompletedEv for the terminal.
void	`setSnapshotEnabled (boolean enabled)` Sets enable/disable status of CiscoTermSnapshotEv. If disabled, CiscoTermSnapshotEv and CiscoTermSnapshotCompletedEv are not sent to applications.
boolean	`getRTPKeyEvEnabled ()` Returns the enable/disable status of CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv.
void	`setRTPKeyEvEnabled (boolean enabled)` Sets enable/disable status for CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv.

CiscoTerminal

void	`createSnapshot () throws InvalidStateException` This method generates CiscoTermSnapshotEv, which contains security statusof current active call on the terminal. To access this method, the terminal must be in CiscoTerminal.IN_SERVICE state, and CiscoTermEvFilter.setSnapshotEnabled () must be set to True.

CiscoMediaTerminal

void	`register (CiscoMediaCapability[] capabilities, int[]supportedAlgorithms)` The CiscoMediaTerminal must be in the CiscoTerminal.UNREGISTERED state and its provider must be in the Provider.IN_SERVICE state. Thisinterface provides dynamic registration with secure media. Ifapplications do not invoke this method, the media gets terminated in non-secure mode.
void	`register (java.net.InetAddress address, int port, CiscoMediaCapability[] capabilities, int[] algorithmIDs)` The CiscoMediaTerminal must be in the CiscoTerminal.UNREGISTERED state, and its provider must be in the Provider.IN_SERVICE state. This interface provides static registration with secure media. If applications do not register this interface, the media remains non-secured. AlgorithmIDs indicate SRTP algorithms that this CTIPort supports. AlgorithmIDs maybe only one of CiscoSupportedAlgorithms.

CiscoRouteTerminal

void	`register (CiscoMediaCapability)[] capabilities, int registrationType, int[] algorithmIDs` The CiscoRouteTerminal must be in the CiscoTerminal.UNREGISTERED state, and its provider must be in the Provider.IN_SERVICE state. By default, media gets terminated in non-secure mode. AlgorithmIDs indicate SRTP algorithms that this CTIPort supports. AlgorithmIDs may be only one of CiscoSupportedAlgorithms.

CiscoSupportedAlgorithm Constants

AES_128_COUNTER

Secured Monitoring and Recording

This feature enables Cisco JTAPI to monitor and record secured calls. Monitoring and recording of calls was introduced in Release 6.0 of the Cisco Unified Communications Manager, but it did not support secured monitoring or recording of calls. For this release, the feature also supports secured calls. With this enhancement a supervisor or recorder can monitor or record a secure call only if its device security capability is same as or more than that of the agent. If the security capability of the monitor initiator's device is less than that of the target, the request for monitor fails. Recording request fails if the recording is attempted for an authenticated device, or if the security capability of the recorder is non-secured and that of the agent is Encrypted.

Cisco JTAPI throws a PriviledgeViolationException with CTIERR_SECURITY_CAPABILITY_MISMATCH, when the monitoring request is rejected due to the supervisor not meeting the security capabilities of the agent. A new API getTransactionID() is added to CiscoTermConnMonitorInitiatorInfoEv and CiscoTermConnMonitorTargetInfoEv.

CiscoJTAPI delivers a new event CiscoAddrMonitoringTerminatedEv when the monitoring session is torn down. This event is delivered to the Supervisor who had started the secured monitoring session but had dropped off from the monitoring call.

New APIs getCiscoAddrMonitoringTerminatedEvFilter() and setCiscoAddrMonitoringTerminatedEvFilter() have been added to the interface CiscoAddrEvFilter for applications to get or set the filter value for the CiscoAddrMonitoringTerminatedEv. By default, the filter is set to True and the event is delivered. To stop receiving this event, applications must set this filter to False.

As before, When a monitoring call (call used by monitor initiator) is conferenced, the final call may not have any connection to monitor target. When monitor initiator conferences another party to a monitoring call, both parties can to listen to the audio between monitor target and caller.

Interface Changes

CiscoJtapiException, on page 410, CiscoTermConnMonitorInitiatorInfoEv, on page 581, CiscoTermConnMonitorTargetInfoEv, on page 583, CiscoAddrMonitorTerminatedEv, on page 282, CiscoAddrEvFilter, on page 299,

Message Sequences

Secured Monitoring Use Cases, on page 1276, Secured Recording, on page 1440

Backward Compatibility

This feature is backward compatible.

SelectRoute Interface Enhancement

The SelectRoute interface gets enhanced to take the parameters PreferredOriginalCalledNumber and PreferredOriginalCalledOption. This enables applications to reset the OriginalCalled value to a specified PreferredOriginalCalledNumber when the call gets routed. This interface takes a list of PreferredOriginalCalledNumber, PreferredOriginalCalledOption, and corresponds them to the RouteSelected list. If the call gets routed to Route at index I in the RouteSelected list, the PreferredOriginalCalledNumber and PreferredOriginalCalledOption at index I get used. Applications get the following behavior with different values for these parameters.

Note Below x, point to the index where the call is being routed. For example, if the call gets routed to Route n, then value of x will equal n. If a PreferredOriginalCalledOption at index x is invalid or out of range, JTAPI defaults it to CiscoRouteSession.DONOT_RESET_ORIGINALCALLED, and if PreferredOriginalCalledOption is null, all the routing gets done with option CiscoRouteSession.DONOT_RESET_ORIGINALCALLED.

When PreferredOriginalCalledOption[x] Is Set to CiscoRouteSession.RESET_ORIGINALCALLED

- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list contains O1, O2, … On, if R1 is available, then call will be routed to R1, and OriginalCalledNumber will be set to O1; if R1 is busy and R2 is available, then call will be routed to R2, and OriginalCalledNumber will be set to O2 … and so on.
- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list contains O1, O2, … Om, and m < n, if R1 is available, the call will be routed to R1, and preferredOriginalCalled will be set to O1; if R1 is busy and R2 is available, the call will be routed to R2, and OriginalCalledNumber will be set to O2 and so on until m. From Route m+1, if Rm+1 is available, the call will be routed to Rm+1, and OriginalCalledNumber will be set to Rm+1, and so on. Lastly, if Rn is available, the call gets routed to Rn, and OriginalCalledNumber gets set to Rn".
- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list is NULL, then if R1 is available, the call will be routed to R1, and OriginalCalledNumber will be set to R1; if R1 is busy and R2 is available, the call will be routed to R2, and OriginalCalledNumber will be set to R2 … and so on.

When PreferredOriginalCalledOption[x] Is Set to CiscoRouteSession.DONOT_RESET_ORIGINALCALLED

- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list contains O1, O2, .. On, the call will be routed to one of the available routes, and the OriginalCalledNumber will remain unchanged.
- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list contains O1, O2, … Om, and m < n, the call will be routed to one of the available routes, and the OriginalCalledNumber will remain unchanged.
- If RouteSelected list contains Routes R1, R2 .. Rn, and preferredOriginalCalled list is NULL, the call will be routed to one of the available routes and OriginalCalledNumber will remain unchanged.

Note When OriginalCalled gets set to PreferredOriginalCalled, LastRedirectingParty number also gets reset to PreferredOriginalCalled.

The following new or changed interfaces exist for SelectRoute Interface Enhancement:

int	**selectRoute** (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] preferredOriginalCalledNumber, int[] preferredOriginalCalledOption) Selects one or more possible destinations for routing a call.

PreferredOriginalCalledOption takes one of the following values:

static int	**DONOT REESET_ORIGINALCALLED** Optional parameter value for PreferredOriginalCalledOption that specifies not to reset OriginalCalled.
static int	**REESET_ORIGINALCALLED** Optional parameter value for PreferredOriginalCalledOption that resets OriginalCalled to preferredOriginalCalledNumber.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for SelectRoute Interface Enhancement, see Message Sequence Charts, on page 755.

selectRoute() with Calling Search Space and Feature Priority

The selectRoute() has feature priority and calling search space parameters as an array. This API provides the flexibility of different feature priorities and calling search spaces for each route selected.

Interface Changes

CiscoRouteSession, on page 517

Message Sequences

selectRoute() with Calling Search Space and Feature Priority, on page 1118

Backward Compatibility

This feature is backward compatible. The selectRoute() API remains functional and interoperates with the overloaded selectRoute() API.

Set MessageWaiting

SetMessageWaiting provides a method for applications to set the message-waiting lamp or indicator for an address. Invoke the method on an address that is in the same partition as the destination.

The following interface specifies whether the message waiting indicator should be activated or deactivated for the address that the **destination** specifies. If **enable** is true, message waiting activates if not already activated. If **enable** is false, message waiting deactivates if not already deactivated.

```
{
public void setMessageWaiting (java.lang.String destination,  boolean enable)
   throws javax.telephony.MethodNotSupportedException,
          javax.telephony.InvalidStateException,
          javax.telephony.PrivilegeViolationException

}
```

Shared Line Support

Shared line represents the same DN appearances on multiple terminals. CiscoJtapi provides support for Shared Line, which provides applications with the ability to control shared DN terminals, hold a call on one shared DN Terminal and unhold the same call from another shared DN Terminal, make calls between two shared lines, initiate a call from one shared line terminal while another active call exists on another shared line terminal with the same DN.

Share line provides the following interfaces:

- CiscoAddress.getInServiceAddrTerminals()—Returns an array of terminals for which the address is in service.

  ```
  Terminal {} getInServiceAddrTerminals();
  ```

- CiscoAddrOutOfService.getTerminal()—Returns the terminal that is going out of service.

  ```
  Terminal getTerminal();
  ```

- CiscoAddrInService.getTerminal()—Returns the terminal that is going in service.

  ```
  Terminal  getTerminal();
  ```

- CiscoConnection.setRequestController(TerminalConnection tc)—Allows an application to select a terminalConnection that is associated with a connection on which you can perform park, redirect, or disconnect operations. You need to do this in a situation where more than one active TerminalConnection exists in a SharedLine scenario.
- CiscoConnection.getRequestController()—Returns TerminalConnection that application sets as request controller.
- CiscoAddrAddedToTerminalEv—Gets sent when the following conditions occur:
  - A Terminal/Device gets added into the user controlList that contains a SharedDN, which sends the event to the application. In other words, if user has an address in control list, and a new device gets added with same address in control list, this event gets sent.
  - An EM (extension mobility) user logs into the terminal with a profile that contains a SharedDN. In this scenario, this event notifies that a new terminal is added to an already existing Address.
  - A new SharedDN is added to a device in a user control list

  Interface getTerminal() returns the terminal that gets added to the address.

  Interface getAddress() returns the address on which a new terminal is added.

- CiscoAddrRemoveFromTerminalEv—Gets sent when the following conditions occur:
  - A user removes a Terminal/Device from the user controlList that contains a SharedDN. In other words, if a user has a shared address in a control list, and one of the devices with same address gets removed, this event gets sent.
  - An EM(extension mobility) user logs out from the terminal that had a profile that contains a SharedDN. This event notifies applications that one of the terminals is removed from an existing Address.
  - A new SharedDN (SharedLine) is removed from a device in a control list.

  Interface getTerminal() returns the terminal that gets removed from the address.

  Interface getAddress() returns the address from where the terminal gets removed.

The following changed or new behaviors exist for a SharedLine:

- Behavior changes for CiscoAddress event include
  - JTAPI applications will receive multiple CiscoAddrInServiceEv for shared line addresses. Applications can use CiscoAddrInServiceEv.getTerminal() to get the terminal on which address goes in service.
  - JTAPI applications receive multiple CiscoAddrOutOfServiceEv for shared line addresses. Applications can use CiscoAddrInServiceEv.getTerminal() to get terminal on which address goes out of service.
  - The address state goes in service when a first shared line goes in service; for example, when the first CiscoAddressInServiceEv gets received.
  - The address state goes out of service when the last shared line goes out of service; for example, when the last CiscoAddressOutOfServiceEv gets received.
- For an incoming call, all the line appearances of a shared line ring. To applications, this gets presented as one active call (callActiveEv), one Connection(ConnCreatedEv), and multiple terminalConnection(TermConnCreatedEv one each for each shared line).
- Calls get presented to all terminals. When a call is in a ringing state, the state of the terminal connection equals Ringing. When a the shared line answers, the terminalConnection state goes to an active state, while other terminalConnections on the shared line go to a passive state, and callControlTerminalConnection for all the shared lines at this point go into a bridged state. When a call is put on hold, all the terminal connections go into an active state, and callControllTerminalConnection goes to a held state. At this point, any terminal can retrieve the call. The retrieving terminal terminalConnection remains in an active state, and callControlTerminalConnection goes to a talking state while all other shared terminals terminalConnections go into a passive state. Simultaneously, CallControlTerminalConnection changes from a held state to a bridged state.
- A shared line can make a call to another shared line of the same DN. In this scenario, the call includes only one connection and multiple terminal connections.
- When a shared line makes a call to another shared line of the same DN, the post condition for this equals only one connection.
- For a shared line connection with two active terminalConnections (such as barge), Connection.Disconnect() does not result in disconnected connection.

If an application is monitoring only a SharedDN Connection with only a passive or bridged TerminalConnection, invoking any API on the connection results in a PreConditionException.

- Similar to the previous scenario, if all the connections of a call monitored by an application have only a Passive or Bridged TerminalConnection, all APIs on the call throw a PreConditionException (such as `Call.Drop()`).
- If more than one active TerminalConnection exists on a shared line, Call.drop() does not return in CallInValid in the following scenarios:
  - A normal two-party call between A and B, where A represents a SharedLine with A' and A' barged into the call

    The application does not monitor A' and B. If the application issues a Call.drop(), the A' TerminalConnection goes into a passive state, but the call does not go InValid.
  - Similar to above, if A, A' , A" and B are in a Conference Call

    The application monitors only A and A', and Call.drop() does not result in the call going InValid. Only the A and A' terminal connections go passive.
  - A, A', and B, B' represent a SharedLine address

    A calls B, B answers, and A' and B' barge into the call. The application monitors only A and B. In this scenario, Call.drop() results in a TerminalConnection of A and B going passive, but the call does not go InValid.
- If a TerminalConnection is in a passive or bridged state or Passive/InUse state, all APIs on the TerminalConnection() throw a PreConditionException. A TerminalConnection only allows an API Terminal ConnectionJoin() (called Barge) in the passive or bridged state. TerminalConnection does not currently support TerminalConnection Join().
- If more than one active or talking TerminalConnections exists in a connection, applications may have to end one before issuing an API on the connection like Redirect(), Park(), Disconnect(). You can select TerminalConnection by using API Connection.setRequestController (TerminalConnection tc).
- If a call gets held on SharedLine terminals and an application issues a Connection.Disconnect (), the applications may set a particular TerminalConnection through API Connection.setRequestController(TerminalConnection tc). If requestcontroller is not set, all HeldTerminalConnections get dropped, and connection goes to a disconnected state. If only one HeldConnection gets dropped, the call remains present on other SharedLines terminals. The call appearance disappears from the dropping terminal, which disallows the terminal from barging into the call or participating in feature operations on the call.

For details on the interface changes, see Cisco Unified JTAPI Extensions, on page 243 To view the message flow for shared lines, see Message Sequence Charts, on page 755

Silent Monitoring

This feature provides the ability to silently monitor calls using an IP Phone. The caller represents the end point, which calls or receives a call from the monitor target. The monitor target is the party to monitor (in a call centre, the agent), and the monitoring party is the monitor initiator (the supervisor).

The recording feature lets applications record conversations on any observed address. Three recording configurations are available:

The silent monitoring feature lets applications listen to a live conversation between two other parties. The monitor initiator cannot talk to either the monitor target or the caller. The feature provides notification tones when legal compliance is required.

Only an application request can initiate monitoring. The application must send a monitor request for each call that it wants to monitor. The system can only monitor calls that are in a connected state. On the successful completion of a monitor request, the audio stream between the monitor target and the caller streams to the monitor initiator. The monitor target receive a tone:

- if the monitor target is configured to receive a tone, or
- if the application requests a tone when it starts the monitor

Applications can monitor calls if they belong to the Standard CTI Allow Call Monitor user group or can be used outside of contact center. The system delivers monitoring-related events to all call observers.

“Monitor” is a reserved word that should not be configured as display names for any lines in the system. Other reserved words are “Conference, ” “Park Number, ” “Barge, ” and “CBarge.”

When a monitoring session is established, the terminal observer on the monitoring initiator receives Cisco RTP events. Although the media for a silent monitoring call flows only in one direction, getMediaConnectionMode() would return CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE instead of CiscoMediaConnectionMode.RECEIVE_ONLY. Applications should expect to find the same behavior in CiscoMediaOpenLogicalChannelEv if a CTIPort is used as the monitor initiator.

When a monitoring call (the call used by the monitor initiator) is conferenced, the final call does not have any connection to the monitor target. When the monitor initiator conferences another party into a monitoring call, both parties can listen to the audio between the monitor target and the caller.

The following interfaces extend TermConnEv and are delivered to the call observer. For shared lines, the system delivers these events to call observers on the address or terminal of the talking terminal connections. Applications receive no events if they have only the terminal whose connection is in the INUSE or BRIDGED state.

CiscoTermConnMonitoringStartEv

`CiscoTermConnMonitoringStartEv`

Indicates the start of monitoring and is delivered to the call observer on the monitor target. Using getMonitorType() on this event returns the monitor type.

CiscoTermConnMonitoringEndEv

`CiscoTermConnMonitoringEndEv`

Indicates the end of monitoring and is delivered to the call observer on the monitor target.

CiscoTermConnMonitorInitiatorInfoEv

Exposes monitor initiator information and is delivered to the call observer of the monitor target. This interface has one method:`CiscoMonitorInitiatorInfo getCiscoMonitorInitiatorInfo ()`

Returns a CiscoMonitorInitiatorInfo that exposes the terminal name and address of the monitor initiator.

CiscoTermConnMonitorTargetInfoEv

Exposes monitor target information and is delivered to the call observer of monitor target. This interface has one method:`CiscoMonitorInitiatorInfo getCiscoMonitorTargetInfo()`

Returns a CiscoMonitorInitiatorInfo that exposes the terminal name and address of the monitor target.

Two new error codes notify applications about monitoring failures:

- CTIERR_PRIMARY_CALL_INVALID is returned by CiscoException.getErrorCode() for exceptions that occur when a monitoring request fails due to the call going idle or getting transferred.
- CTIERR_PRIMARY_CALL_STATE_INVALID is returned when the monitoring request fails due to the call transitioning to a different state where monitoring cannot be invoked.

This release introduces a new AddressType, MONITORING_TARGET. JTAPI creates a connection on an address of this type for a monitoring target address; CiscoAddress.getType() returns this value.

Backward Compatibility

This feature is backward compatible. Applications will not see any new events unless this feature is configured and used on one of the application-controlled addresses. The administrator can enable this feature by adding Standard CTI Allow Call Monitor user groups.

For detailed information about these interface changes, see the following topics:

- CiscoJtapiException, on page 410
- Related Documentation, on page 283
- CiscoCall, on page 326
- CiscoMediaTerminal, on page 448
- CiscoMonitorTargetInfo, on page 460
- CiscoMonitorInitiatorInfo, on page 459
- CiscoProvider, on page 486
- CiscoProviderCapabilities, on page 498
- CiscoProviderCapabilityChangedEv, on page 500
- CiscoRecorderInfo, on page 505
- CiscoTerminalConnection, on page 630
- CiscoTermConnMonitorInitiatorInfoEv, on page 581
- CiscoTermConnMonitorTargetInfoEv, on page 583

Secured Monitoring

With this enhancement a supervisor can monitor a secure call only if its device security capability is same as or more than that of the agent. If the security capability of the monitor initiator's device is less than that of the target, the request for monitor fails.

Cisco JTAPI throws a PriviledgeViolationException with CTIERR_SECURITY_CAPABILITY_MISMATCH, when the monitoring request is rejected due to the supervisor not meeting the security capabilities of the agent. A new API getTransactionID() is added to CiscoTermConnMonitorInitiatorInfoEv and CiscoTermConnMonitorTargetInfoEv.

CiscoJTAPI delivers a new event CiscoAddrMonitoringTerminatedEv when the monitoring session is torn down. This event is delivered to the Supervisor who had started the secured monitoring session but had dropped off from the monitoring call.

The APIs getCiscoAddrMonitoringTerminatedEvFilter() and setCiscoAddrMonitoringTerminatedEvFilter() have been added to the interface CiscoAddrEvFilter for applications to get or set the filter value for the CiscoAddrMonitoringTerminatedEv. By default, the filter is set to True and the event is delivered. To stop receiving this event, applications must set this filter to False. As before, When a monitoring call (call used by monitor initiator) is conferenced, the final call may not have any connection to monitor target. When monitor initiator conferences another party to a monitoring call, both parties can to listen to the audio between monitor target and caller.

Secured Monitoring Interface Changes

CiscoJtapiException, on page 410, CiscoTermConnMonitorInitiatorInfoEv, on page 581, CiscoTermConnMonitorTargetInfoEv, on page 583, CiscoAddrMonitorTerminatedEv, on page 282, CiscoAddrEvFilter, on page 299

Message Sequences

Secured Monitoring Use Cases, on page 1276, Secured Recording, on page 1440

Backward Compatibility

This feature is backward compatible.

Single Sign-On

The Single Sign-On feature allows Cisco JTAPI applications to use the single sign-on ticket to authenticate instead of a user ID and password.

Applications fetch the service ticket for the OpenSSO server from the active directory and then pass the ticket to Cisco JTAPI in the string used in the getProivder(String str) API. Applications can set the single sign-on ticket as ssoticket = "ssotokenfromat".

Only end users can use this feature.

Applications using this feature need not specify the user ID and password in the getProvider string.

If an application is used by an end user and has the Standard CTI Secure Connection role enabled, then a user ID is required in the provider string. No password is required.

This solution is designed around an active directory with a Kerberos environment to achieve Windows desktop Single Sign-On. If an active directory with a Kerberos environment is unavailable, then an alternate equivalent setup is available, which includes a KDC, an authentication server, and a domain controller.

Sample Code

```
String ssoticket  =  getSSOticket(); //application implementation
String providerString  =  cucmserver + ssoticket +";";
JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
try
{
  Provider provider  =  peer.getProvider (providerString);
}
catch (Exception exp )
```

```
{
  if ( exp instanceof PlatformException)
  {
    switch (((CiscoJtapiException)exp).getErrorCode() )
    {
    case CiscoJtapiException. CTIERR_SSO_DISABLED:
      System.out.println("SSO feature not enabled on CUCM ");
      break;
    case CiscoJtapiException. CTIERR_SSO_AUTH_SERVER_DOWN:
      System.out.println("server down");
      break;
    }
  }
  else
  {
    System.out.println("Exception  =  " + exp.toString());
  }

}
```

SSO Cookie

JTAPI supports authentication using SSO Cookie from Release 10.0.1 and later. An SSO Cookie, once generated, is valid for the entire session. The cookie can be reused during that session. SSO Cookie is supported only on a Secure Connection. Cisco JTAPI does not allow authentication using SSO Cookie over non-secure connections.

Applications must also provide the fully qualified name of the client and server certificates in the providerString.

The following new keywords are being introduced to be used in the provider string : **ssocookie**, **cCert**, **sCert**.

The providerString must be in the following format when using an SSO Cookie:

providerString = "ssocookie = <cookie>;cCert = <fully qualified client certificate>;sCert = <fully qualified server certificate>;"

Interface Changes

See CiscoJtapiException, on page 410

Message Sequences

See Single Sign-On, on page 1468

Backward Compatibility

This feature is backward compatible.

Single Step Transfer

This interface allows applications to transfer a call to an address. Cisco Unified JTAPI continues to support this interface as defined in JTAPI 1.2 specification, but the events that are delivered to applications are changed from the previous versions of Cisco Unified JTAPI.

In previous versions of Cisco Unified JTAPI, the original call goes to a held state, and a new call gets created between the transfer controller and destination when applications use this interface. After successful completion of transfer, both calls on transfer controller go to an IDLE state. If a transfer fails, the original call remains in

a held state, and applications retrieve the call. CiscoTransferStart and end events get delivered to the applications at the start and completion of the transfer operation.

Applications get the following changes:

- A new call does not get created.
- CiscoTransferStartEv and CiscoTransferEndEv do not get delivered to applications.
- The state of the original call is retained if the transfer operation fails.

The pre and post conditions of this interface did not change.

To view the message flow for Single Step Transfer, see Message Sequence Charts, on page 755

SIP 3XX Redirection

The SIP Redirect server receives SIP requests and responds with 3xx(redirection) responses, which direct the client to contact an alternate set of SIP addresses. This enhancement supports the Cisco Unified Communications Manager Redirection (3xx) Call Control primitive in compliance with RFC 3261. The Cisco Unified Communications Manager Redirection primitive processes SIP 3xx responses and does sequential hunting to each contact address from the 3xx response. Cisco Unified Communications Manager Redirection primitive also handles feature interactions that result from performing this operation. Cisco Unified JTAPI exposes new reason codes in all CallEvs, which indicate when connection and terminalConnection are created and destroyed as a result of this primitive.

LastRedirectAddress may change if feature interactions like JTAPI Redirect or CallForwardNoAnswer occur when the Redirection primitive is hunting for a target. If the target does not answer and Cisco Unified Communications Manager Redirect takes control of the call to send it to next target, lastRedirectAddress is set to the party who originally sent the SIP 3xx response.

If a diversion header is present in the SIP 3xx response, the 3xx primitive uses the first value of the diversion header for lastRedirectParty, and JTAPI applications will see the diversion header element as lastRedirectAddress.

To maintain backward compatibility, JTAPI exposes the new API CiscoCallEv.getCiscoFeatureReason() in the CiscoCallEv interface, which contains the reason as CM_REDIRECTION.

Note Applications should be aware that new feature-specific reason codes could be returned from this API, and applications should provide default behavior for unrecognized reason codes.

The following sections describe the interface changes for SIP 3XX Redirection.

Public Interface CiscoFeatureReason

static int	`REASON_CM_REDIRECTION` This reason indicates that event is a result of 3xx response from the CM_REDIRECTION primitive in Cisco Unified Communications Manager.

CiscoCallEv

int	`getCiscoFeatureReason()` A feature specific reason for this event. Applications should make sure to handle unrecognized reasons and provide default behavior as this interface may not be backward compatible as new reasons might be added in the future.

SIP Phone Support

This release of Cisco Unified Communications Manager allows phones that run SIP to register and interoperate with phones that run SCCP. The following sections describe the new interfaces introduced to support phones that run SIP along with the limitations and differences in behavior with respect to phones that support SCCP. Though not all existing features are supported on phones that run SIP, the general behavior in terms of JTAPI events and interfaces for phones that run SIP are similar to that of a phone that runs SCCP.

JTAPI applications can only control Cisco Unified IP Phone 7900 Series that run SIP, which includes Cisco Unified IP 7970 phones. Applications should not include Cisco Unified IP 7960, 7940, and other phones that run the SIP protocol in their control list. JTAPI applications cannot control third-party phones that run SIP, so third-party phones that run SIP should not be included in the control list.

In prior releases, JTAPI supported an initial feature set on phones that run SIP. In this release support is added for the following functionality on phones that run SIP:

- Park for Phones that run SIP
- Unpark Phones that run SIP

Tip The order of events for consult calls differs for phones that run SIP and SCCP phones. Consider the following scenario:

1. Terminal A initiates a call to the shared line B/B'.
2. The shared line initiates a consult call to Terminal C.

   If the shared line is a SIP device, the following call events occur:

   - B (active) receives: OnHold -> Select -> NewCall
   - B' (remote-in-use) receives: Select -> NewCall -> OnHold

   However, if the shared line is a SCCP device, the call events are Select -> OnHold -> NewCall on both terminals.

   If the application is only monitoring, call.getConsultingTerminalConnection() may return null.

JTAPI supports the following features for phones that run SIP:

- Call.connect; offhook
- answer; disconnect; drop; hold, unhold
- consult; transfer; conference; redirect

- playdtmf, deviceData

JTAPI supports the following events for phones that run SIP:

- CiscoTermDeviceStateEv, RTP events, inService, and OutOfService
- MediaTermConnDtmfEv (only out of band is supported), transfer start and end events, conference start and end events, CiscoToneChangedEv, and CiscoTermConnPrivacyChangedEv

Behavior of phones that run SIP differ from that of phones that run SCCP in the following ways:

- Call Rejection—When a call is made to a phone that runs SIP, the phone can choose to reject the call. In this case, applications perceive CallActive, ConnCreatedEv followed by ConnDisconnectedEv for the address on the SIP terminal. This is similar to RP rejecting the call.
- Consult without media calls involving SIP phones should be transferred within 1.5 seconds after the call is connected.
- For phones that run SIP, enbloc dialing is always used even if the user first goes off hook before dialing digits. The phone waits until all the digits are collected before sending the digits to the Cisco Unified Communications Manager . This means that CallCtlConnDialingEv is delivered only after enough digits are pressed on the phone to match one of the configured dialing patterns.
- Applications should configure “out of band DTMF” on all devices to receive MediaTermConnDtmfEv.

Events for CTI ports, route points, and phones that run SCCP are not changed.

When a Cisco Unified IP Phone 7900 Series model that runs SIP using UDP as transport fails connectivity with Cisco Unified Communications Manager , JTAPI applications receive the events CiscoTermOutOfServiceEv and CiscoAddrOutOfServiceEv for the terminal and address defined for the phone. Because of the inherent delay in UDP in detecting the connectivity loss, the Cisco Unified IP Phone 7900 Series that runs SIP may visually show as registered after applications have already been notified with the out-of-service events.

If Cisco Unified IP Phone s 7960, 7940, and non-Cisco Unified IP Phone 7900 Series that run SIP are included in the control list, exceptions are thrown when observers (both observer and call observers) are added to the address or terminal and CiscoTermRestrictedEv is delivered to a provider observer. The cause for these events would be CiscoRestrictedEv.CAUSE_UNSUPPORTED_PROTOCOL.

CiscoTerminal exposes new interface getProtocol() to indicate whether terminal is a phone that runs SCCP or a phone that runs SIP. CiscoTerminalProtocol defines the values that are returned by getProtocol().

The following new interfaces that are defined on CiscoCall let applications get URL information for external SIP entities.

Public Interface CiscoCall

CiscoPartyInfo	`getLastRedirectingPartyInfo()`
CiscoPartyInfo	`getCurrentCallingPartyInfo()`
CiscoPartyInfo	`getCurrentCalledPartyInfo()`
CiscoPartyInfo	`getCalledPartyInfo()`

Public Interface CiscoPartyInfo

CiscoUrlInfo	`getUrlInfo()`
Address	`getAddress()`
string	`getDisplayName()`
string	`getUnicodeDisplayName()`
boolean	`getAddressPI()`
boolean	`getDisplayNamePI()`
boolean	`getlocale()`

Public Interface CiscoUrlInfo

int	`getUrlType()`
	`Final int URL_TYPE_TEL`
	`Final int URL_TYPE_SIP`
	`Final int URL_TYPE_UNKNOWN`

string	`getHost()`
string	`getUser()`
int	`getPort()`

int	`getTransportType()`
	`Final int TRANSPORT_TYPE_UDP`
	`Final int TRANSPORT_TYPE_TCP`

Public Interface CiscoTerminal

int	**`getProtocol ()`**

CiscoTerminalProtocol

static int	`PROTOCOL_NONE` Indicates an unrecognized or unknown protocol type
static int	`PROTOCOL_SCCP` Indicates the device is using SCCP to communicate to Cisco Unified Communications Manager

static int	PROTOCOL_SIP Indicates the device is using SIP to communicate to Cisco Unified Communications Manager

SIP REFER or REPLACE

REFER is a SIP method that is defined by RFC 3515. The REFER method indicates that the recipient (referee, identified by the Request-URI) should contact a third party (referred to as the target) by using the contact information that is provided in the request. This REFER method allows the party who is sending the REFER (referrer) to be notified of the outcome of the referenced request.

Cisco Unified Communications Manager, being a Back-To-Back User Agent (B2BUA), processes both inside and outside dialog inbound REFER on behalf of the Referee. As result of REFER, Cisco Unified Communications Manager creates a call between the Referee and the Refer-to-Target. Ifthere is a previously existing call between the Referrer and the Referee, the call at the Referrer gets dropped after REFER completes.

The REPLACES feature is the replacement of an existing SIP dialog with a new dialog. A SIP dialog is a call between two SIP user agents; a Cisco Unified Communications Manager dialog is a half call (callleg). The REPLACES feature is triggered either by REFER or by an INVITE. Cisco Unified Communications Manager handles a REPLACES request on behalf of the recipient of the REPLACES header. The request is associated with a new dialog and the requesting party is the party that wants to replace another party in the existing dialog (call) identified in the REPLACES header. Cisco Unified Communications Manager disconnects the dialog (call) identified in the REPLACES header and connects the requesting party.

JTAPI is enhanced to model Call events caused by the Cisco Unified Communications Manager REFER and REPLACE features in the JTAPI call model. JTAPI provides applications with the capability to handle call events caused by REFER and REPLACE features. JTAPI does not provide any interface for applications to initiate REFER or REFER/INVITE with REPLACES requests; however, JTAPI can handle the call events properly.

These two features are backward compatible. JTAPI provides events that are caused by REFER/REPLACE with CAUSE_NORMAL. Applications can get feature-specific reasons from the new interface CiscoCallEv.getCiscoFeatureReason().

Note This interface provides feature-specific reasons for current and new features, but this method will not remain backward compatible in future releases. Applications using this interface must implement default handling to avoid future backward-compatibility issues.

The following sections describe the interface changes for SIP REFER/REPLACE.

CAUSE Provided for REFER/REPLACE

JTAPI provides CAUSE_NORMAL for events that caused by REFER/REPLACES. Applications should use CiscoCallEv.getCiscoFeatureReason() to get the feature-specific reason.

Interface Provided on CiscoCallEv

This interface provides CiscoFeatureReason in the JTAPI call event. Older features, such as transfer, continue to receive the old CiscoCause that is provided by the previous interface, CiscoCallEv.getCiscoCause(). This new interface provides REASON_TRANSFER for transfer.

```
com.cisco.jtapi.extensions
Interface CiscoCallEv
```

int	**`getCiscoFeatureReason()`** This interface returns Cisco Unified Communications Manager Feature Reason.

Interface CiscoFeatureReason

JTAPI provides CiscoFeatureReason in Call events caused by features. CiscoFeatureReason is provided for existing as well as new Cisco Unified Communications Manager features. For REFER and REPLACES features, the reason would be REASON_REFER and REASON_REPLACES. This interface will provide new reasons for any new features that may be introduced in the future, and is not backward compatible.

Applications using CiscoFeatureReason should expect to receive new reasons in later releases and must implement default behavior to maintain the Application's backward compatibility.

Applications that use CiscoFeatureReason should expect to receive new reasons in later releases and must implement default behavior to maintain backward-compatibility.

Public Interface CiscoFeatureReason

static int	`REASON_REFER` Reason returned for events that are sent for REFER by Cisco Unified Communications Manager.
static int	`REASON_REPLACE` Reason returned for events that are sent for REPLACE by Cisco Unified Communications Manager.

SIP Trunk Early Offer

The SIP Trunk Early Offer feature allows the SIP trunk to support early offer outbound calls. The SIP trunk does not use a Media Termination Point (MTP) when the media capabilities and port information of the phone is available.

If the media port information is not available, the Cisco Unified Communications Manager allocates an MTP to provide an offer.

If the application enables this feature and makes a call that goes though a SIP trunk, the Cisco Unified Communications Manager must have the IP address and the port information of the registered terminal even before the media is established. This eliminates the need for MTP.

The following are the changes done from the JTAPI perspective:

- A new interface, CiscoBaseMediaTerminal, extends CiscoTerminal.

- A new register() API has the following arguments:
  - IP Address
  - Port
  - Media Capability
  - Algorithm ID
  - IP_V6 Address
  - Addressing Mode
  - Registration Type

Applications use register() API to register CiscoMediaTerminal and CiscoRouteTerminal with the following registration types available in CiscoBaseMediaTerminal.

- CiscoBaseMediaTerminal.NO_MEDIA_REGISTRATION (applicable only for route points)
- CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION (for dynamic registration of CTI ports and route points)
- CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT
- CiscoBaseMediaTerminal.STATIC_MEDIA_REGISTRATION (for static registration of CTI port)
- CiscoBaseMediaTerminal.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT

Note The applications use the register() APIs on CiscoRouteTerminal and CiscoMediaTerminal for route points and CTI ports to specify the registration type.

To enable this feature, select one of the following:

- CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT for registration type to register a CTI port or a route point dynamically
- CiscoBaseMediaTermial.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT for registration type to register a CTI port or a route point statically.

If an application has enabled this feature and initiated a call that goes through a SIP Trunk, CiscoJTAPI delivers a new event CiscoMediaOpenIPPortEv. On recieving this event, applications query for the registration type using the API getRegistrationType(), which is exposed on this interface, and do the following based on the value returned.

- If return value is CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT, applications must set the RTP Parameters and open the port. At present, the applications set the RTP parameters upon receiving CiscoMediaOpenLogicalChannelEv for dynamically registered CiscoMediaTerminal and CiscoRouteTerminal.
- If return value is CiscoBaseMediaTerminal.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT,

applications must open the port. At present, most of the applcations open statically registered terminals when they receive RTP events.

If an application tries to register a terminal, which is already registered with registration type as CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT or CiscoBaseMediaTerminal.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT, with a different registration type, JTAPI throws a PlatformException with the error code as CiscoJtapiException.CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC_GETPORT_SUPPORT or CiscoJtapiException.CTIERR_MEDIA_ALREADY_TERMINATED_STATIC_GETPORT_SUPPORT, respectively.

A new API, isRTPRequired(), is also exposed on the interface CiscoMediaOpenLogicalChannelEv to indicate if the applications must set the RTP parameters or not when they receive this event.

Applications must check the API when they recieve the CiscoMediaOpenLogicalChannelEv and set the RTP Parameters only when the return value is true.

Note Early offer is not supported for IPv6 calls in release 8.5(1).

If an application registers a terminal with registration type as CiscoBaseMediaTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT or CiscoBaseMediaTerminal.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT and the IP addressing mode as IPv6, the registration follows but this feature does not come into effect. The applications do not receive the CiscoMediaOpenIPPortEv.

The application must close the ports when it receives media termination events or when a call is disconnected.

When IPv6 support is added, the application receives two CiscoMediaOpenIPPortEv, for dual mode devices, one for IPv4 and the other for IPv6 addresses. When the call is answered, application closes the unused port based on MediaIPAddressingType in CiscoMediaOpenLogicalChannelEv.

The service parameter, Fail Call Over SIP Trunk if MTP Allocation Fails, decides if the call must go through as a delayed offer or not. If applications do not set the RTP parameters when they receive CiscoMediaOpenIPPortEv for a dynamically registered terminal with get port support, this service parameter decides if the call must go through as a delayed offer or not.

Interface Changes

See CiscoBaseMediaTerminal, on page 323, CiscoMediaOpenIPPortEv, on page 441, CiscoMediaOpenLogicalChannelEv, on page 443, CiscoJtapiException, on page 410

Message Sequences

See SIP Trunk Early Offer, on page 1492

Backward Compatibilty

This feature is backward compatible.

This feature is applicable only when applications register the CiscoMediaTerminals and CiscoRouteTerminals with registrationType as CiscoTerminal. DYNAMIC_MEDIA_REGISTRATION_GET_PORT or CiscoBaseMediaTerminal. STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT.

Star (*) 50 Update

The Star (*) 50 feature enables you to divert a call to original called party (value returned by CiscoCall.getCalledAddress() method) and the called party (value returned by CiscoCall.getCurrentCalledAddress() method) from phone UI. After pressing the iDivert softkey, a menu displays that identifies the names of the original called party and the called party.

The user selects one of the two names and the call is redirected to the voice mailbox of the selected party. With the legacy iDivert, the call is diverted to original called party voice mailbox by just pressing iDivert softkey. Cisco Unified Communications Manager Administration introduced the following Service parameters to configure this feature:

- iDivert Legacy Behavior—Determines whether the phone uses the legacy iDivert behavior when a user presses the iDivert softkey or the enhanced *50 iDivert behavior. If the iDivert legacy service parameter is set to true, the iDivert legacy behavior is adopted and vice versa.
- Allow QSIG during iDivert–Determines whether iDivert legacy is allowed in deployments that have voice messaging integration over QSIG trunks and only used when the Use Legacy iDivert service parameter is set to true.
- iDivert User Response timer–Determines the number of seconds that Cisco Unified Communications Manager Administration waits for a response from the user before the iDivert screen is removed. If no user action occurs by the time this timer expires, the screen is removed from the phone. If the Use Legacy iDivert service parameter is set to true, Cisco Unified Communications Manager Administration ignores this parameter.

There is no interface change at JTAPI layer for this feature. The behavior changes from JTAPI application point of view means that Calls could either go to voice mail of OrigicalCalled Party or Called.

Backward Compatibility

This feature is backward compatible.

Super Provider (Disable Device Validation)

When a JTAPI application user is configured, the system administrator normally associates a certain set of terminals (Cisco Unified IP Phones and devices) with this application user, who can control and monitor only this set of terminals. The Super Provider feature gives applications the ability to control and monitor any terminal in a Cisco Unified Communications Manager cluster.

The new createTerminal() new interface in CiscoProvider lets the application create a terminal by specifying a terminalName. JTAPI does not provide the capability to get the terminalName through any interface. The CiscoProvider.createTerminal(terminalName) returns the terminal. If the terminal already exists in the provider domain, JTAPI returns the existing terminal.

A second new interface, CiscoProvider.deleteTerminal(), lets the application delete the CiscoTerminal objects that are created by using the CiscoProvider.createTerminal() interface. If the terminal object does not exist or the application did not create the terminal with the CiscoProvider.createTerminal() interface, JTAPI throws exceptions.

JTAPI also provides a new interface on CiscoProviderCapabilities, canObserveAnyTerminal(), which can be enabled for application users through Cisco Unified Communications Manager Administration user

configuration. Applications can use this interface to determine whether they have sufficient capability to invoke the createTerminal(terminalName) interface. If the application does not have sufficient capability and this interface is invoked, JTAPI throws a PrivilegeViolationException. If the application provides a terminalName that does not exist in the Cisco Unified Communications Manager cluster, JTAPI throws a InvalidArgumentException.

Superprovider and Change Notification

Superprovider enhancements for JTAPI in this release consist primarily of the following changes.

When the "Superprovider privilege" gets disabled from Cisco Unified Communications Manager Administration after a provider opens, JTAPI gets notified through a CTI Change Notification Event and cleans up all the devices that it has opened that are not in its control list.

JTAPI informs applications about the change using the "CiscoProviderCapabilityChangedEvent." This new event gets issued when the flag changes and indicates whether the flag has been enabled or disabled. When a device that is not in the control list is opened in the Superprovider mode, then moved to the control list, JTAPI moves the device into its control list.

- When a normal application receives a "CiscoProviderCapabilityChangedEvent" with the flag set, it means the Superprovider privilege has been granted to it, and it can start acquiring devices not in its control list.
- When a Superprovider application receives a "CiscoProviderCapabilityChangedEvent" with the Superprovider flag not set, it means that the Superprovider privilege has been removed for it. The following sequence of events then occurs:
  - Applications receive a Provider OOS event and all devices acquired/opened by it are closed.
  - Applications receive a CiscoTermRemovedEv for all devices not in the control list that have been acquired or opened.
  - Applications receive a Provider inService event when JTAPI succeeds in reconnecting to CTI as a normal user.
  - Applications receive device and line information.
  - Applications receive CiscoTermCreatedEv for all controlled devices that were open before the provider went OOS.
- JTAPI notifies applications by using the "CiscoProviderCapabilityChangedEvent" when the "park DN monitoring" flag is changed from Cisco Unified Communications Manager Administration.
  - When an application receives this event with the flag set, it does a register feature for the controlling park DN.
  - When an application receives this event with the flag not set, JTAPI again informs applications by using a "CiscoProviderCapabilityChangedEvent" and closes all the park DN addresses.
- JTAPI notifies applications by using the CiscoProviderCapabilityChangedEvent" when the "change calling party number" flag is changed from Cisco Unified Communications Manager Administration.
  - When an application receives this event with the flag set, it can change the calling party number.
  - When an application receives this event with the flag not set, it cannot change the calling party number.

Applications should not change the calling party number when this flag is disabled.

- When a device that is not in the control list is opened or acquired by Superprovider, and is then deleted from Cisco Unified Communications Manager Administration, JTAPI closes the terminal object and sends a CiscoTermRemovedEvent to the application for that device.

Interface Changes

As a part of the Superprovider and change notification enhancements, JTAPI exposes the following API to applications. The JTAPI implementation for Superprovider and the handling of certain Provider capabilities has changed as a result. Superprovider enhancements for JTAPI in this release consist of the JTAPI QBE interface, changes in JTAPI behavior, and the new API which is exposed to applications.

JTAPI delivers CiscoProviderCapabilityChangedEv to the applications, with the following format. Applications should be able to receive and process this new event from JTAPI.

```
public interface CiscoProviderCapabilityChangedEv {
    public CiscoProviderCapabilities getCapability ();
}
```

CiscoProviderCapabilities have the following new methods for setting calling party modify privilege for the provider:

```
  public boolean canModifyCallingParty();
  public void setCanModifyCallingParty(boolean value);
```

CiscoProviderCapabilityChangedEv is delivered to the applications with the appropriate flag values.

After this, the following sequence of events occurs:

- JTAPI sends provider OOS events to the application and device/line OOS to devices and lines in the control list that are open.
- JTAPI then tries to reconnect to CTI.
  - If reconnect succeeds, JTAPI sends a provider inService event and reopens all the devices in the control list that were previously open.
  - If reconnect does not succeed, JTAPI shuts down the provider and sends a ProviderClosedEvent.
- If Superprovider privilege is added, JTAPI sends a CiscoProviderCapabilityChangedEv to the applications with the appropriate flag values.
  - If the MonitorParkDN flag is enabled, JTAPI sends a CiscoProviderCapabilityChangedEv with the monitor park DN flag set to true.
  - If the MonitorParkDN flag is disabled, JTAPI sends a CiscoProviderCapabilityChangedEv with the monitor park DN flag set to false.

    JTAPI also closes all the park DN addresses and delivers a CiscoAddrRemovedEv to applications.
- When the ModifyCgPn flag is changed, JTAPI sets a flag in the provider object that is checked during redirect scenarios, and applications are accordingly allowed or denied permission to change the calling party.

  JTAPI also delivers a CiscoProviderCapabilityChangedEv with the flag set to modify CgPn.

CiscoProvider Interface

boolean	`hasSuperproviderChanged()` Tells the application whether the Superprovider privilege changed.
boolean	`hasModifyCallingPartyChanged()` Tells the application whether the ModifyCgPn privilege changed.
boolean	`hasMonitorParkDNChanged()` Tells the application whether the Park DN monitoring privilege changed.

Backward Compatibility

This feature is not backward compatible.

Support for Cisco Unified IP Phone 6901

Cisco Unified IP Phone 6901 is a new IP phone with keypad similar to other basic Cisco IP phones but this phone does not have display, speaker phone, or head set jack. This phone supports only SCCP protocol. Features such as Park, Unpark, Call Pickup, Group Call Pickup, Direct Transfer, Call Forward All, and Join are not supported as softkeys are not provided for these features. These features are supported only from Cisco Unified Communication Manager. Cisco Unified IP Phone 6901 is a one line device and can support two calls per line. So, features such as Join Across Lines and Direct Transfer Across Lines cannot be supported by these devices.

One of the limitations of this phone is that to intiate or answer a call, the phone must be off-hook. If the phone is on-hook and the user initiates or answers a call, JTAPI throws InvalidStateException to the application with error code as CiscoJtapiException.OPERATION_NOT_AVAILABLE_IN_CURRENT_STATE.

Another limitation is that Cisco Unified IP Phone 6901 does not accept XSI objects from applications, but if the application calls sendData() API for these phones, JTAPI throws an exception for the request to the application with the error code as CiscoJtapiException.COMMAND_NOT_IMPLEMENTED_ON_DEVICE.

Table 6: List of Supported or Unsupported Features on Cisco Unified IP Phone 6901

Feature	Supported/Unsupported	Scope
Park	Supported	From application only
UnPark	Supported	From application only
CallPickup	Supported	From application only
Hold/Retrieve	Supported	From phone and application
DirectTransfer	Supported	From phone and application
sendData() API	Unsupported	
JoinAcrossLines	Unsupported	As only one line can be configured on the phone

Feature	Supported/Unsupported	Scope
DirectTransferAcrossLines	Unsupported	As only one line can be configured on the phone
AutoBarge	Supported	From phone only
Recording	Unsupported	BIB cannot be configured on the phone
Monitoring	Supported	From application only **Note** If 6901device is a supervisor. If it is an agent then monitoring is not supported.
Hunt-list support	Supported	
Conference	Supported	From phone and application.
CallForwardAll	Supported	From application only.
Redirect	Supported	From application only.
EM-Login	Unsupported	
Intercom	Unsupported	Intercom line cannot be configured

Interface Changes

See CiscoJtapiException, on page 410

Message Sequences

See Support for Cisco Unified IP Phone 6901, on page 1506

Backward Compatibility

This feature is backward compatible.

Support for Cisco Unified IP Phone 6900 Series

This feature allows Cisco Unified JTAPI applications to control terminals with rollover mode enabled. In rollover mode, terminals are configured with multiple addresses with the same DN but in different partitions or with different DNs. When rollover mode is enabled, consult calls can be created on the next available address on the terminal. Cisco Unified IP Phone 6900 Series can be configured with rollover mode.

A new role Standard CTI Allow Control of phones supporting rollover mode has also been introduced to allow applications to control terminals with rollover enabled. Applications that support this new behavior where consult calls are created on a different address, must include this role to their application or end user. If not, all terminals configured with rollover mode are restricted and exceptions are thrown to addObserver() requests.

Applications that support this behavior are must add call observer on the terminal or add call observers on all addresses on the terminal. Since consult call is created on the next available addresses, exceptions are thrown to consult requests if call observers are not added to all addresses.

Join across lines must be enabled on Cisco Unified IP Phone 6900 Series to successfully complete conferences from applications.

Cisco Unified Communications Manager Release 8.6, JTAPI supports multiple calls per line configuration on Cisco Unified IP Phone 69xx series. Prior to Release 8.6, Cisco Unified IP Phone 69xx series supported only one call per line, where Maximum Number of Calls/Busy Trigger defined for a line (MNC/BT) cannot exceed 2/1. With multiple calls per line, Cisco Unified IP Phone 69xx series supports more than one call per line, and MNC/BT is configured to values greater than 2/1.

Outbound Rollover Behavior for 69xx Phones

With MNC/BT configured as 2/1,

When a second call is initiated from a line, the new call will be created on (rollover to) the second line. Cisco Unified Communications Manager Release 8.6 supports outbound rollover. If MNC is greater than 2, there can be multiple calls on the line before the rollover occurs. For both Cisco Unified Communications Manager Release 8.5 and 8.6, the outbound rollover occurs if MNC-1 calls are active on the line.

Outbound Rollover is supported only on the endpoints. Using the JTAPI application, you can make MNC calls for a line; however, rollover will not happen at MNC-1, even if a second line exists).

Interface Changes

See CiscoProviderCapabilities, on page 498 and CiscoProviderCapabilityChangedEv, on page 500

Message Sequences

See Support for Cisco Unified IP Phone 6900 Series, on page 1230

Backward Compatibility

This feature is backward compatible.

Support for 100+ Directory Numbers

This feature enables users to have more than 100 Directory Numbers associated with a Device (Phones, CTI Ports and Route Points). JTAPI supports this feature and displays the corresponding addresses on the terminal to the application.

Interface Changes

There are no interface changes.

Message Sequences

There are no message sequences.

Backward Compatibility

This feature is backward compatible.

Support for VMware

From Cisco Unified Communications Manager Release 8.0(1), Cisco JTAPI can be used on VMware ESXi version 4.0. The application can use Windows 2003 and Windows 2008 virtual machines on the above VMware version to run Cisco JTAPI. For more information on the supported Java Virtual Machines, see the following table.

Table 7: Supported JVM Versions for Cisco Unified Communications Manager

Operating System	Version	Unified CM 10.0	Unified CM 10.5	Unified CM 11.0	Unified CM 11.5	Unified CM 12.0	Unified CM 12.5
Linux	AS 3.0	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Linux	RHEL 7 (64 bit)	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Linux	RHEL 3.7	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Linux	RHEL (32 bit)	RH 5.5 Sun JVM 1.6.0.29	RH 5.5 Oracle JVM 1.7.0.40	RH 5.5 Oracle JVM 1.7.0.76	RH 5.5 Oracle JVM 1.7.0.79	RH 5.5 Oracle JVM 1.7.0.79	RH 5.5 Oracle JVM 1.7.0.79
Linux	RHEL 5.5 (64 bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Linux	RHEL 6 (64 bit)	Sun JVM 1.7.0.40	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Solaris	6.2 on Sparc and x86	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Windows	Windows XP 2003, 2008 Server(32-bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Not supported	Not supported
Windows	Vista (32 bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Not supported	Not supported
Windows	Windows 7(32 and 64 bit) 2008 Server R2(64 bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Windows	Windows 8(32 and 64 bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Windows	Windows Server 2012 R1 (32 bit)	Sun JVM 1.6.0.29	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Windows	Windows 8.1(32 and 64 bit)	Not supported	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79

Operating System	Version	Unified CM 10.0	Unified CM 10.5	Unified CM 11.0	Unified CM 11.5	Unified CM 12.0	Unified CM 12.5
Windows	Windows Server 2012 R2 (64 bit)	Sun JVM 1.7.40	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Windows	Windows 10(32 and 64 bit)	Not supported	Oracle JVM 1.7.0.40	Oracle JVM 1.7.0.76	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79
Windows	Windows Server 2016(64 bit)	Not supported	Not supported	Not supported	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79	Oracle JVM 1.7.0.79

Interface Changes

There are no interface changes.

Message Sequences

There are no message sequences.

Backward Compatibility

Not applicable.

Swap or Cancel and Transfer or Conference Behavior

This feature enables Cisco Unified JTAPI support for Swap and Cancel operations on supported IP phones.

When a Swap operation is invoked, it puts an active call on hold and retrieves the held call. When a Cancel operation is invoked, it breaks the consulting relationship between primary and consulting calls. These operations can only be invoked from supported phones. The Cisco Unified JTAPI interface does not allow SWAP/CANCEL operations to be invoked from the application. Whenever a user presses the SWAP key on a phone, JTAPI delivers CallCtlTermConnHeldEv and CallCtlTermConnTalkingEv for active and held calls and indicates their state change with CiscoFeatureReason.REASON_NORMAL.

When a CANCEL operation is invoked and the relationship is broken between primary and consulting calls, Cisco Unified JTAPI is still able to use the Direct Transfer or Join feature to complete the transfer or conference operation. If the user presses the CANCEL key on phone after initiating a consult, the conference or transfer is not completed. Pressing the CANCEL key on phone triggers a Cancel notification to the application; Cisco Unified JTAPI sends CiscoCallFeatureCancelledEv to indicate the CANCEL operation. CiscoCallFeatureCancelledEv.getConsultCall() returns the earlier created consult call.

When the CANCEL operation is performed during a connected transfer or conference, the following can occur:

- The user presses the CANCEL key before selecting the Active Call softkey:

  In this case, pressing the Transfer key creates a consultCall GC3, and pressing the CANCEL key triggers CiscoCallFeatureCancelledEv on GC2 with GC3 as a consult call.

- The user presses the CANCEL key after pressing the Active Calls softkey but does this before selecting the call on phone UI.

  In this case, pressing the Active Calls softkey on the phone UI makes consultCall GC3 IDLE, but there is no CANCEL notification, as other feature operations are possible. However, if the user presses CANCEL, the CiscoCallConsultCancelEv with consult call as null, is trigerred.

- The user presses the Active Call softkey, selects a call and then presses CANCEL.

  In this case, the selected call is returned as a consultCall with CiscoCallFeatureCancelledEv.

Interface Changes

See CiscoCallFeatureCancelledEv, on page 359

Message Sequences

See Swap or Cancel and Transfer or Conference Behavior Change, on page 1168

Backward Compatibility

This feature is backward compatible.

For this release, the Swap or Cancel feature is enabled without a service parameter to turn it off. This means that Cisco Unified JTAPI always supports or reports events for Swap or Cancel for phones which support this feature.

However, to provide backward compatibility for applications, a new permission that enables control of these devices and to enable SWAP or CANCEL operation has been added. A new standard role Standard Supports Connected Xfer/Conf and a standard user group are added in the admin pages for this feature. Applications can control these devices only if this new role is associated to the application user, assuming that the application uses JTAPI client 7.1.2 or higher. So, by default these devices are listed as restricted and only if application upgrades to handle this feature and associates the new permission can it control these devices. If the application uses an older JTAPI client the devices are not restricted but if the application tries to observe these devices (which supports this feature to be invoked manually) then JTAPI throws an exception and marks these devices as restricted from there on.

Since, the feature is designed to provide an enhanced user experience, it is strongly recommended for all Cisco Unified JTAPI applications to evaluate and support this feature and upgrade if necessary with the code logic to handle the old behavior and the new behavior.

Terminal and Address Capability Settings

This feature introduces interfaces that expose different configuration settings of address and terminal. These interfaces can be called even when the terminal or address is in out-of-service state. All interfaces return the values that are configured while registering with Cisco Unified Communications Manager. If the terminal is not registered, an InvalidStateException is returned. Application can get the voice mail pilot even if the terminal is not registered.

All the other changes, except voice mail, require the terminal to be reset for the new values to take effect. Interfaces return new values only after phones re-register after reset. Applications can use the interface CiscoProvTerminalRegisteredEv to read the configuration of the terminal and address.

The following configurations are exposed on CiscoAddress:

- Max calls configured
- Busy Trigger
- Position of address on a terminal
- Voice mail pilot
- ASCII and Unicode labels

CiscoTerminal provides new interfaces to applications to get the following configurations of a terminal:

- IPV4 and IPV6 IP addresses
- Outbound Rollover configuration

Terminal and address capability feature introduces new interfaces to determine if the terminal is capable of performing the following features:

- Consult call rollover
- Out bound call rollover
- Join across lines
- Direct transfer across lines
- Join on same line
- Direct transfer on same line

Interface Changes

See Related Documentation, on page 283, CiscoAddrEvFilter, on page 299, CiscoAddrVoiceMailPilotChangedEv, on page 320, CiscoTerminal, on page 611, CiscoProvFeatureID, on page 479, CiscoProvTerminalRegisteredEv, on page 484, and CiscoProvTerminalUnRegisteredEv, on page 485.

Message Sequences

See Terminal and Address Capability Settings Use Cases, on page 1234

Backward Compatibility

This feature is backward compatible.

Terminal and Address Restrictions

This enhancement restricts applications from controlling and monitoring a certain set of terminals and addresses when the administrator configures them as restricted in Cisco Unified Communications Manager Administration.

The administrator can configure a particular line on a device (address on a particular terminal) as restricted. If a terminal is added into the restricted list in Cisco Unified Communications Manager Administration, all addresses on that terminal are also marked as restricted in JTAPI. If an application comes up after the configuration is completed, it can know whether a particular terminal or address is restricted from checking the interface CiscoTerminal.isRestricted() and CiscoAddress.isRestricted(Terminal). For shared lines,

applications can query the interface CiscoAddress.getRestrictedAddrTerminals(), which indicates whether an address is restricted on any terminals.

If a line (address on a terminal) is added into the restricted list after an application comes up, the applications will see CiscoAddrRestrictedEv. If the address has any observers, applications will see CiscoAddrOutOfService. When a line is removed from the restricted list, applications will see CiscoAddrActivatedEv. If an address has any observers, applications see CiscoAddrInServiceEv. Ifan application tries to add observers on an address after it is restricted, a PlatformException gets thrown. However, if any observers are added before the address is restricted, they will remain as is, but applications cannot get any events on these observers unless the address is removed from the restricted list. Applications can also choose to remove observers from an address.

If a device (terminal) is added to the restricted list after an application comes up, the application will see CiscoTermRestrictedEv. If the terminal has any observers, the application will see CiscoTermOutOfService. If a terminal is added to the restricted list, JTAPI also restricts all addresses that belong to that terminal and applications will see CiscoAddrRestrictedEv. If a terminal is removed from the restricted list, applications will see CiscoTermActivatedEv and CiscoAddrActivatedEv for the corresponding addresses. If an application tries to add observers on a terminal after it is added to the restricted list, a PlatformException is thrown. However, if observers are added before the terminal is restricted, they remain as is, but applications cannot get any events on these observers unless the terminal is removed from the restricted list.

If a shared line is added to the restricted list after an application comes up, the application will see CiscoAddrRestrictedOnTerminalEv. If any address observers exist on the address, the application will see CiscoAddrOutOfServiceEv for that terminal. If all shared lines are added to the restricted list, when the last one is added, applications will see CiscoAddrRestrictedEv. If a shared line is removed from the restricted list after the application comes up, applications will see CiscoAddrActivatedOnTerminalEv. If any observers exist on the address, the application will see CiscoAddrInServiceEv for that terminal. Ifall shared lines in the control list are removed from the restricted list, applications will see CiscoAddrActivatedEv when the last one is removed, and all addresses on terminals will receive InService events.

If all shared lines in the control list are marked as restricted, and an application tries to add observers, a platform exception is thrown. If a few shared lines are in the restricted list, while others are not, when an application adds an observer on the address. Only non-restricted lines go in service.

If any active calls are present when an address or terminal is added to the restricted list and reset, applications will see connection and TerminalConnections get disconnected.

If no addresses or terminals are added to the restricted list, this feature is backward compatible with earlier versions of JTAPI: no new events are delivered to applications.

The following sections describe the interface changes for address and terminal restrictions.

CiscoTerminal

boolean	`isRestricted()` Indicates whether a terminal is restricted. If the terminal is restricted, all associated addresses on this terminal are also restricted. Returns true if the terminal is restricted; returns false if it is not restricted.

CiscoAddress

javax.telephony.Terminal[]	`getRestrictedAddrTerminals()` Returns an array of terminals on which this address is restricted. If none are restricted, this method returns null. In shared lines, a few lines on terminals may be restricted. This method returns all the terminals on which this particular address is restricted. Applications cannot see any call events for restricted lines. If a restricted line is involved in a call with any other control device, an external connection gets created for the restricted line.
boolean	**`isRestricted`**`(javax.telephony.Terminal terminal)` Returns true if any address on this terminal is restricted. Returns false if no addresses on this terminal are restricted.

```
public interface CiscoRestrictedEv extends CiscoProvEv {
 public static final int ID  =  com.cisco.jtapi.CiscoEventID.CiscoRestrictedEv;

 /**
  * The following define the cause codes for restricted events
  */

public final static int CAUSE_USER_RESTRICTED  =  1;

public final static int CAUSE_UNSUPPORTED_PROTOCOL  =  2;

}
```

This is the base class for restricted events and defines the cause codes for all restricted events. CAUSE_USER_RESTRICTED indicates the terminal or address is marked as restricted. CAUSE_UNSUPPORTED_PROTOCOL indicates that the device in the control list is using a protocol that is not supported by Cisco Unified JTAPI. Existing Cisco Unified IP 7960 and 7940 phones that are running SIP fall in this category.

CiscoAddrRestrictedEv

Public interface **CiscoAddrRestrictedEv** extends CiscoRestrictedEv. Applications will see this event when a line or an associated device is designated as restricted from Cisco Unified Communications Manager Administration. For restricted lines, the address goes out of service and does not come back in service until it is activated again. If an address is restricted, addCallObserver and addObserver throws an exception. For shared lines, if a few shared lines are restricted, and others are not, no exception is thrown, but restricted shared lines do not receive any events. If all shared lines are restricted, an exception is thrown when adding observers. If an address is restricted after adding observers, applications see CiscoAddrOutOfServiceEv, and when the address is activated, the address goes in service.

CiscoAddrActivatedEv

Public interface **CiscoAddrActivatedEv** extends CiscoProvEv. Applications see this event whenever a line or an associated device is in the control list and is removed from the restricted list in the Cisco Unified Communications Manager Administration. If any observers exist on the address, applications see CiscoAddrInServiceEv. If no observers exist, applications can try to add observers, and the address goes in service.

CiscoAddrRestrictedOnTerminalEv

Public interface **CiscoAddrRestrictedOnTerminalEv** extends CiscoRestrictedEv. If a user has a shared address in the control list, and if one of the lines is added into the restricted list, this event is sent. Interface getTerminal() returns the terminal on which the address is restricted. Interface getAddress() returns the address that is restricted.

javax.telephony.Address	`getAddress()`
javax.telephony.Terminal	`getTerminal()`

CiscoAddrActivatedOnTerminal

Public interface **CiscoAddrActivatedOnTerminalEv** extends CiscoProvEv. When a shared line or a device that has a shared line is removed from the restricted list, this event will be sent. The interface getTerminal() returns the terminal that is being added to the address. The interface getAddress() returns the address on which the new terminal is added.

javax.telephony.Address	`getAddress()`
javax.telephony.Terminal	`getTerminal()`

CiscoTermRestrictedEv

Public interface **CiscoTermRestrictedEv** extends CiscoRestrictedEv. Applications see this event when a device is added into restricted list from Cisco Unified Communications Manager Administration after the application launches. Applications cannot see events for restricted terminals or addresses on those terminals. If a terminal is restricted when it is in InService state, applications get this event and terminal and corresponding addresses move to the out-of-service state.

CiscoTermActivatedEv

Public interface **CiscoTermActivatedEv** extends CiscoRestrictedEv.

javax.telephony.Terminal	`getTerminal()` Returns the terminal that is activated and is removed from the restricted list.

CiscoOutOfServiceEv

static int	`CAUSE_DEVICE_RESTRICTED` Indicates whether an event is sent because a device is restricted.
static int	`CAUSE_LINE_RESTRICTED` Indicates whether an event is sent because a line is restricted.

CiscoCallEv

static int	`CAUSE_DEVICE_RESTRICTED` Indicates whether an event is sent because a device is restricted.
static int	`CAUSE_LINE_RESTRICTED` Indicates whether an event is sent because a line is restricted.

SHA-512 Support for Digital Signatures

From Release 11.5(1), Cisco Unified Communications Manager supports the SHA-512 algorithm for CTL, ITL and TFTP configuration file encryption. The **TFTP File Signature Algorithm** enterprise parameter has been added to allow administrators to select which encryption algorithm will be used. By default, this enterprise parameter is set to SHA-1, but you can reconfigure the parameter to SHA-512.

Backward Compatibility

The SHA-512 algorithm is not supported prior to release 11.5(1). If an application is running a Cisco JTAPI version that is prior to 11.5(1), that application must be using the SHA-1 algorithm in order to maintain a secure connection.

Use Cases

SHA Support for Digital Signatures, on page 1529

Transfer

The transfer feature moves the participants of one call, the transferred call, to another call, the final call. Moving participants in a call trasitions their associated connections to the DISCONNECTED state in the transferred call and new connections for these participants getting created in the final call. Similarly, any associated TerminalConnections transition into the DROPPED state in the transferred call and get created in the final call. Cisco extensions by definition mark the start and the end of the events that relate to transfer.

You can correlate the newly created connection objects with the old connection objects by use of the CiscoConnection.getConnectionID() method to obtain the CiscoConnectionID for the old and new connections. Matching connections possess identical CiscoConnectionID objects when you compare them by using the CiscoConnectionID.equals() method.

CiscoTransferStartEv

This event indicates that the transfer operation started, and the events that follow relate to this operation. Specifically, Connections and TerminalConnections get both removed and added as a result of the transfer.

Applications may obtain the two calls that are involved in transfer-transferred call and final call and the transfer controller information from this event. If the JTAPI application is not observing the transfer controller, the transfer controller information does not get made available in this event.

CiscoTransferEndEv

This event indicates that the transfer operation ended. After this event is received, the application can assume that all involved parties transferred and that all Connections and TerminalConnections moved to the final call.

Transfer Scenarios

In the following scenarios, A, B, and C represent three parties that are involved in the transfer.

Consult Transfer; B Is the Transfer Controller

In a consult transfer, applications can redirect calls to a different address, and the transferrer can "consult" with the transfer destination before redirecting.

- A calls B on call Call1.
- B answers and consults to C on call Call2.
- B transfers call Call2 to call Call1.

To do this type of transfer, use the following JTAPI methods:

- Call2.setTransferEnable(true) (This optional method means that transfer is enabled in the call object by default.)
- Call2.consult(TermConnB, C)
- Call1.transfer(Call2)

Note During consult transfer, Call1.transfer(Call2) will transfer the call but not Call2.transfer(Call1).

The following table lists the core events that observers of A and B receive between the CiscoTransferStartEv and the CiscoTransferEndEv.

Table 8: Core Events for Observers of A and B

Meta Event Cause	Call	Event	Fields
META_UNKNOWN	Call1	CiscoTransferStartEv	transferredCall = Call2 finalCall = CalltransferController = TermConnB
META_CALL_TRANSFERRING	Call1	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	

Meta Event Cause	Call	Event	Fields
META_CALL_TRANSFERRING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv CCallInvalidEv C	
META_UNKNOWN	Call2	CallObservationEndedEv	
META_UNKNOWN	Call1	CiscoTransferEndEv	transferredCall = Call2 FinalCall = Call1 transferController = TermConnB

Arbitrary Transfer; A Is the Transfer Controller

In an arbitrary transfer, one call can get transferred to another call, irrespective of how either call was created. Unlike consult transfer, no need exists to first create one of the calls by using the consult method.

- A calls B on call Call1.
- A puts Call1 on hold.
- A calls C on call Call2.
- A transfers Call1 to Call2.

To do this type of transfer, use the following JTAPI methods:

- Call2.transfer(Call1) to transfer call Call1 to final call Call2, or
- Call1.transfer(Call2) to transfer call Call2 to final call Call1

Assuming Call1.transfer(Call2) was called, the following table lists the core events that observers on A and C receive between CiscoTransferStartEv and CiscoTransferEndEv.

Table 9: Core Events for Observers of A and C

Meta Event Cause	Call	Event	Fields
META_UNKNOWN	Call1	CiscoTransferStartEv	transferredCall = Call2 finalCall = Call1 transferController = TermConnB

Meta Event Cause	Call	Event	Fields
META_CALL_TRANSFERRING	Call1	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv C CallInvalidEv C	

Transfer and Conference Extensions

You may find that transfer and conference events are difficult to understand in JTAPI. This happens because, when the participants are moved from one call to the other, JTAPI represents this action by deleting the parties from one call and adding them to the other call. It may confuse you for an application to receive an indication that a party dropped from the call when, in reality, it is in the process of being moved. The Cisco Unified JTAPI implementation defines some extra events that make it easier for applications to deal with these functions.

Transfer and DirectTransfer

The transfer feature provides the ability to transfer a call.

The direct transfer feature represents the ability to transfer any of the two calls that are present on the line, so controller of the call drops out, and other two parties remain active on the call. This functionality gets supported with one enhancement: this feature can be done in any state of the call and also can be redesigned to work with new CTI events. The following enhancements apply to the transfer feature:

- The application can transfer two held calls.
- The application can have OneHeld and OneConnected call in any order.

- The application can transfer any two calls that are present on the line.

The following changed or new interfaces exist for Transfer and DirectTransfer:

- CiscoTransferStarted. getTransferControllers()—This new interface, which is provided for SharedLine scenarios, supports multiple terminalConnections if a SharedLine is a TransferController. When a transferController is not a SharedLine, only a TerminalConnection occurs in the list. This method returns null if the transfer controller is not being observed.
- CiscoTransferStarted. getTransferController()—This current interface, which behaves as it does for a normal transfer, may exhibit a different behavior for SharedLines. When a transferController is a SharedLine, multiple TerminalConnections exist. This method returns an ACTIVE TerminalConnection; however, if the application is not observing the ACTIVE TerminalConnection, this method returns one of the PASSIVE TerminalConnections.
- CiscoTransferEnded isSuccess()—This new interface, which is provided for the CiscoTransferEnded event, returns true if the transfer operation succeeds and false if the transfer fails. Transfer failure may result from the following events:
  - The party dropped the call before CallProcessing could complete the transfer.
  - CallProcessing cannot Complete the transfer.

The following changed or new behaviors exist for JTAPI Transfer:

- No Hold or UnHold messages occur with an arbitrary transfer.
- If a precondition for a transfer request has been modified, an application can issue transfer in any state of the call.
- If an application does not have an active TerminalConnection that is passed as an argument, Call.consult() throws a PreConditionException/InvalidArgumentException.
- If controller does not have any active TerminalConnection, Call.Transfer() throws a PreconditionException/InvalidArgumentException.

To view the message flow for Transfer and DirectTransfer, see Message Sequence Charts, on page 755

Translation Pattern Support

If a calling party transformation mask is configured for a translation pattern that is applied to a JTAPI application-controlled Address, the application may recognize extra connections that are created and disconnected when both the calling and called party are observed. A Connection is created for a transformed calling party instead of the actual calling party and CiscoCall.getCurrentCallingParty() would return the transformed calling party, when only the called party is observed. In general, JTAPI might not be able to create the appropriate Connection in the Call, and might not be able to provide correct information for currentCalling, currentCalled, calling, called, and lastRedirecting parties.

For example, consider a translation pattern X that is configured with a calling party transformation mask Y and called party transformation mask B. If A calls X, the call goes to B. In this scenario:

- If the application is observing only B, JTAPI creates a Connection for Y and B, and CiscoCall.getCurrentCallingParty() would return Address Y.

- If the application is observing both A and B, a Connection for A and B gets created, a Connection for Y gets temporarily created and dropped, and CiscoCall.getCurrentCallingParty() would return Address Y.

Other inconsistencies in the calling information could occur if further features get performed on a basic call. Cisco recommends that you not configure a calling party transformation mast for a translation pattern that might get applied to JTAPI application-controlled addresses.

Transport Layer Security (TLS)

This feature lets JTAPI applications communicate with CTIManager through a secure connection. CTIManager runs a TLS listener socket to accept connections from JTAPI. Establishing a TLS connection requires a client certificate, which the server uses to authenticate the client, and a server certificate, which the client uses to authenticate the server.

In the Cisco Unified Communications Manager environment, the server certificate exists in the form of CTL on the TFTP server, and JTAPI downloads this certificate. The initial download of CTL is trusted and occurs without verification, so Cisco strongly recommends performing this download in a secure environment. One of the two System Administrator Security Tokens (SAST) that are present in the CTL file signs the CTL; subsequent CTL downloads get verified with the SAST from the old CTL file.

JTAPI connects to CAPF by using the CAPF protocol to get the client certificate (LSC). You can authenticate these certificates with the issuers certificate present in CTL.

CTI tracks the number of provider connections that are created per client certificate. Applications can create only one provider by using a client certificate. If more than one instance of a provider is created, both providers get disconnected from CTI and go out of service. JTAPI will retry the connection to CTI to bring the original provider in service; however, if both instances of provider continue to exist, after a certain number of retries, provider gets permanently shut down, and the client certificate is marked as compromised. Any further attempt to create a provider by using this client certificate fails. Applications must contact the administrator to configure a new instanceId and download a new client certificate to resume operation.

Note Each client certificate is associated with a unique instanceId configured in the Cisco Unified Communications Manager database. Applications can provide an instanceId in providerString as an optional parameter to use a unique certificate while creating a CiscoProvider.

To run multiple instances of applications with TLS, ensure that the application user is configured in the Cisco Unified Communications Manager database with multiple instanceIDs. Applications use these unique instanceIDs to get unique client certificates for each instance.

The JTAPI preferences application provides a graphic user interface to configure the Security parameters and update server/client certificates. Application users need to configure the TFTPServer IP address, CAPFServer IP address, Username, InstanceID, and AuthorizationString parameters through the JTAPI preferences to download/install certificates on the application server.

New interfaces are provided for JTAPI client applications on the client layer object. For example, a JTAPI client interface is provided on the CTIClientProperties class.

This feature is backward compatible with previous releases as JTAPI Applications can still connect to CTIManager on non-secure socket connections.

The following sections describe the interfacr changes for TLS support in JTAPI.

CiscoJtapiPeer.getProvider()

```
public javax.telephony.Provider getProvider(java.lang.String providerString)
throws javax.telephony.ProviderUnavailableException
```

This modified interface takes a new optional parameter InstanceID. It returns an instance of a Provider object, given a string argument that contains the desired service name.

Optional arguments may also be provided in this string, with the following format:

```
< service name > ; arg1  =  val1; arg2  =  val2; ...
```

Where < service name > is not optional, and each optional argument = value pair that follows is separated by a semicolon. The keys for these arguments are implementation-specific, except for two standard-defined keys:

- login—Provides the login user name to the Provider.
- passwd—Provides a password to the Provider.

CiscoJtapiPeer in providerString expects a new optional argument:

- InstanceID—Provides InstanceID for Application Instance.

InstanceID is needed when two or more instances of an application want to connect to Provider (CTIManager) through a TLS connection from the same client machine. Each instance of an application requires its own unique X.509 certificate to establish a TLS connection. If JTAPI attempts to open more that one connection with same username/instanceID, CTIManager rejects the TLS connection. If instanceID is not provided, JTAPI randomly picks one of the instances of USER and, in that case, the connection may fail if a connection for the selected Instance already exists.

If the argument is null, this method returns some default provider as determined by the JtapiPeer object. The returned provider is in the Provider.OUT_OF_SERVICE state.

Post-conditions:

```
this.getProvider().getState()  =  Provider.OUT_OF_SERVICE
```

Specified by

`getProvider` in interface `javax.telephony.JtapiPeer`

Parameters

`providerString` The name of the desired service plus an optional argument.

Returns

An instance of the Provider object.

Throws

`javax.telephony.ProviderUnavailableException`

Indicates that a provider that corresponds to the given string is unavailable.

CiscoJtapiProperties

JTAPI provides an interface on CiscoJtapiProperties to enable or disable the security option and install the client/server certificates that are required to establish a secure TLS socket connection.

```
com.cisco.jtapi.extensions
Interface CiscoJtapiProperties
```

getSecurityPropertyForInstance

```
public java.util.Hashtable getSecurityPropertyForInstance()
```

This interface returns a Hash table with all the parameters set for User/InstanceID. The Hash table gets set with the following “key–value” pairs:

KEY	VALUE
“user”	userName
string “instanceID”	InstanceID
string “AuthCode”	authCode
string “CAPF”	capfServer IP-Address
string “CAPFPort”	capfServer IP-Address port
string “TFTP”	tftpServer IP-Address
string “TFTPPort”	tftpServer IP-Address port
string “CertPath”	certificate Path
string “securityOption”	Boolean security option(true for enable/ false for disabled)
string “certificateStatus”	Boolean certificate status(true for updated/ false for not updated)

Returns—Hash table in the format described previously for the first user and instance.

getSecurityPropertyForInstance

```
public java.util.Hashtable getSecurityPropertyForInstance
(java.lang.String user,  java.lang.String instanceID)
```

This interface returns a Hash table with all the parameters set for User/InstanceID. The Hash table is set with the following “key–value” pairs:

KEY	VALUE
“user”	userName
string “instanceID”	InstanceID
string “AuthCode”	authCode
string “CAPF”	capfServer IP-Address
string “CAPFPort”	capfServer IP-Address port
string “TFTP”	tftpServer IP-Address
string “TFTPPort”	tftpServer IP-Address port
string “CertPath”	certificate Path
string “securityOption”	Boolean security option(true for enable/ false for disabled)
string “certificateStatus”	Boolean certificate status(true for updated/ false for not updated)

Parameters:

`user` - UserName for which you want security parameters

`instanceID` - InstanceID for which you want security parameters

Returns—Hash table in preceding format.

setSecurityPropertyForInstance

```
public void setSecurityPropertyForInstance(java.lang.String user,
java.lang.String instanceID,
                      java.lang.String authCode,
                      java.lang.String tftp,
                      java.lang.String tftpPort,
                      java.lang.String capf,
                      java.lang.String capfPort,
                      java.lang.String certPath,
                      boolean securityOption)
```

You can use this interface to set security properties for the following parameters:

Parameters:

`user`—UserName for which the security parameter is being updated

`instanceID`—InstanceID for which the security parameter is being updated

`authCode`—Authorization string

`capf`—IP-Address of CAPF server

`capfPort`—IP-Address port number on which the CAPF server is running, as defined in a CallManager Service parameter. If the value is null, the default value is 3804.

`tftp`—IP-Address of TFTP server

`tftpPort`—IP-Address port number on which the TFTP server is running. The Cisco Unified Communications Manager TFTP server usually runs on port 69. If the value is null, the default value is 69.

`certPath`—Path where certificate needs to be installed

updateCertificate

```
public void updateCertificate(java.lang.String user,
               java.lang.String instanceID,
               java.lang.String authcode,
               java.lang.String ccmTFTPAddress,
               java.lang.String ccmTFTPPort,
               java.lang.String ccmCAPFAddress,
               java.lang.String ccmCAPFPort,
               java.lang.String certificatePath)
```

This interface installs an X.509 client certificate for the USER instance in the certificate store by connecting to the Cisco Unified Communications Manager Certificate Authority Proxy Function (CAPF) server. Italso downloads the Certificate Trust List (CTL) from the Cisco Unified Communications Manager TFTP server.

If the user credentials are not valid, this method throws a PrivilegeViolationException. If the TFTP server or CAPF server address is not correct, this method throws an InvalidArgumentException. Every instance of an application requires a unique client certificate. If a multiple instanceID is configured in the Cisco Unified Communications Manager database, applications can call this interface multiple times to install a client certificate for every instance.

Pre-conditions—When calling this interface, an application should have network connectivity with the Cisco Unified Communications Manager CAPF and TFTP servers.

Post-conditions—This process installs client and server certificates on the JTAPI application machine.

Parameters:

`user`—Name of the CTI application user that is configured in the Cisco Unified Communications Manager database

`instanceID`—Application instance ID that is configured in the Cisco Unified Communications Manager database. Everyinstance of an application requires a unique ID.

`authCode`—Authorization string that is configured in the Cisco Unified Communications Manager database. You can use the authCode only once for getting certificates.

`ccmTFTPAddress`—IP-Address of the Cisco Unified Communications Manager TFTP server.

`ccmTFTPPort`—IP-Address port number on which the Cisco Unified Communications Manager TFTP server is running. The Cisco Unified Communications Manager TFTP server usually runs on port 69. Ifnull, the default value is 69.

`ccmCAPFAddress`—IP address of the Cisco Unified Communications Manager CAPF server.

`ccmCAPFPort`—Port number on which the Cisco Unified Communications Manager CAPF server is running, as defined in the Cisco Unified Communications Manager Service parameters. If the value is null, the default value is 3804.

`certificatePath`—Directory path where the certificate needs to be installed

Throws:

`InvalidArgumentException`—This exception gets thrown for an invalid TFTP server or CAPF server address.

`PrivilegeViolationException`—This exception gets thrown for an invalid user, instanceID, or authCode.

IsCertificateUpdated

```
public boolean IsCertificateUpdated
      (java.lang.String user,  java.lang.String instanceID)
```

This interface provides information about whether client and server certificates are updated for a given user/instanceID.

Parameters:

`user`—UserName as defined in the Cisco Unified Communications Manager Administration.

`instanceID`—InstanceID for the specified UserName.

Returns—True if certificates are already updated; false if certificates are not updated.

updateServerCertificate

```
public void updateServerCertificate(java.lang.String ccmTFTPAddress,
                  java.lang.String ccmTFTPPort,
                  java.lang.String ccmCAPFAddress,
                  java.lang.String ccmCAPFPort,
                  java.lang.String certificatePath)
```

This interface installs an X.509 server certificate that is given the certificate path. If the TFTP server address is not correct, this method throws an InvalidArgumentException. Auto update applications should use this interface to update the server certificate before invoking an HTTPS connection with Cisco Unified Communications Manager.

Pre-conditions—When calling this interface, applications should have network connectivity with the TFTP server.

Post-conditions—This interface installs the server certificate on the JTAPI application machine.

Parameters:

`ccmTFTPAddress`—IP address of the Cisco CallManager TFTP server.

`ccmTFTPPort`—Port number on which the Cisco Unified Communications Manager TFTP server is running.

If null, the default value is 69.

certificatePath—Directory path for installing the certificate.

`ccmCAPFAddress`—IP address of the Cisco Unified Communications Manager CAPF server.

`ccmCAPFPort`—Port number on which the Cisco Unified Communications Manager CAPF server is running.

If the value is null, the default value is 3804.

Throws:

`InvalidArgumentException`—If the TFTP server address is invalid.

Interface Provided on JTAPI Preferences

The JTAPI Preferences dialog box includes a Security tab to let application users configure the username, instanceId, authCode, TFTP IP address, TFTP port, CAPF IP server address, CAPF server port, and certificate path, and enable secure connection.

- "CAPF server port" number defaults to 3804.

  You can configure this value in the Cisco Unified Communications Manager Administration service parameters window. The CAPF server port value entered through JTAPI Preferences should match the one that is configured in Cisco Unified Communications Manager Administration.

- "TFTP server port" number defaults to 69.

  Do not change this value unless you are advised to do so by the System Administrator.

- "Certificate Path" is where the application wants the sever and client certificates to be installed.

  If this field is left blank, the certificates get installed in the ClassPath of JTAPI.jar.

- "Certificate update Status" provides information on whether a certificate has been updated or not.

- You must select "Enable Secure Connection" to enable a secure TLS connection to Cisco Unified Communications Manager.

  If "Enable Security Connection" is not selected, JTAPI makes a non-secure connection to CTI even if the certificate is updated/installed.

- The "Enable Security Tracing" check box lets you enable or disable tracing for the certificate installation operation.

  If tracing is enabled, you can select three different levels, Error, Debug, or Detailed, from the drop-down menu.

You can use the JTAPI Preference UI to configure a security profile for one or more than one userName/instanceID pair. When application users revisit this window, and have previously configured security profile for a userName/instanceID pair, the security profile automatically gets populated when the user enters a username/instanceID and clicks on other edit box.

The Trace Levels tab in the JTAPI Preferences UI is renamed as JTAPI Tracing. This highlights the fact that the JTAPI Tracing tab only lets you change trace setting for the JTAPI layer. Tracing for the installation of Security certificates must be enabled on the Security tab.

Unicode Support

Cisco Unified Communications Manager release 5.0 supports unicode display names on unicode-enabled IP phones. You can configure ASCII names and unicode names for display names. JTAPI receives all names in unicode and ASCII formats and provides two new interfaces, getCurrentCalledPartyUnicodeDisplayName and getCurrentCallingPartyUnicodeDisplayName, toallow applications to get display names in unicode. Italso provides the ability to get unicode display names during call progress.

JTAPI receives the encoding capability of application controlled IP phones in device registered and device in service events from CTI, locale and language group information in device info response, and provides interfaces to applications to get the locale, alternate script, and unicode capability of IP phones. CiscoTerminal and CiscoTermInServiceEv interfaces are enhanced to provide this information for phones that are in the application control list when the CiscoTerminal is in the inservice state.

JTAPI receives the alternate script information of all parties in the call and provides interfaces to applications to get the language group of the current calling and current called party. Two interfaces, getCurrentCallingPartyLanguageGroup and getCurrentCalledPartyLanguageGroup, are available on CiscoCall to get this information. Applications also receive both ASCII and UCS-2 encoded unicode display names for the current calling and called addresses.

Unicode support for JTAPI also includes:

- CiscoCall interface changes
- CiscoLocales interface changes
- CiscoTerminal / CiscoTerminalInServiceEv interface changes

Applications might need to reconfigure their username/password after upgrading to Release 5.0.

The following sections describe the interface changes for unicode support.

Interface CiscoCall Changes

The following new methods on CiscoCall let applications get the unicode display names and the corresponding locales.

```
/**
 * This interface returns the unicode display name of the current called party
 * in the call.
 */
public String getCurrentCalledPartyUnicodeDisplayName();

/**
 * This interface returns the locale of the current called party unicode
 * display name. CiscoLocales interface lists the supported locales.
 */
public int getCurrentCalledPartyUnicodeDisplayNamelocale();

/**
 * This interface returns the unicode display name of the current calling party
 * in the call.
 */
```

```
public String getCurrentCallingPartyUnicodeDisplayName ();

/**
 * This interface returns the locale of the current called party
 * unicode display name
 */
public int getCurrentCallingPartyUnicodeDisplayNamelocale();
```

CiscoLocales

The CiscoLocales interface lists all the locales that Cisco Unified JTAPI supports.

Note For a list of all supported locales in the most recent release, see the man page for CiscoLocales, on page 436.

```
public interface CiscoLocales
{
  public static final int LOCALE_ENGLISH_UNITED_STATES;
  public static final int LOCALE_FRENCH_FRANCE;
  public static final int LOCALE_GERMAN_GERMANY;
  public static final int LOCALE_RUSSIAN_RUSSIA ;
  public static final int LOCALE_SPANISH_SPAIN ;
  public static final int LOCALE_ITALIAN_ITALY ;
  public static final int LOCALE_DUTCH_NETHERLAND ;
  public static final int LOCALE_NORWEGIAN_NORWAY ;
  public static final int LOCALE_PORTUGUESE_PORTUGAL;
  public static final int LOCALE_SWEDISH_SWEDEN ;
  public static final int LOCALE_DANISH_DENMARK
  public static final int LOCALE_JAPANESE_JAPAN;
  public static final int LOCALE_HUNGARIAN_HUNGARY ;
  public static final int LOCALE_POLISH_POLAND ;
  public static final int LOCALE_GREEK_GREECE ;
  public static final int LOCALE_TRADITIONAL_CHINESE_CHINA;
  public static final int LOCALE_SIMPLIFIED_CHINESE_CHINA;
  public static final int LOCALE_KOREAN_KOREA;
}
```

CiscoTerminalInServiceEv Interface

int	`getLocale()` This method returns the current locale information for this terminal.
int	`getSupportedEncoding ()` This method returns true if this terminal supports unicode.

CiscoTerminal Interface

int	`getLocale()` This method returns the current locale information for this terminal. The CiscoTerminal must be in the CiscoTerminal.IN_SERVICE state to access this method.
int	`getSupportedEncoding ()` This method returns the unicode capability of this Terminal. The CiscoTerminal must be in the CiscoTerminal.IN_SERVICE state to access this method.

The getSupportedEncoding () returns one of the following results that are defined in CiscoTerminal.

```
/**
* Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
* for this Terminal is UNKNOWN
  */
 public final static int UNKNOWN_ENCODING  =  0;
/**
* Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
* for this is NOT_APPLICABLE.
* This is valid for only CiscoMediaTerminals and RoutePoints
*/
 public final static int NOT_APPLICABLE  =  1;
/**
* Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE> for this
* Terminal is ASCII and this terminal supports only ASCII_ENCODING
*/
 public final static int ASCII_ENCODING  =  2;
/**
* Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
* for this Terminal is UCS2UNICODE_ENCODING
*/
 public final static int UCS2UNICODE_ENCODING  =  3;
```

Unrestricted Unified CM

Cisco Unified JTAPI provides support for Unrestricted Cisco Unified Communications Manager, where encryption is disabled.

This feature was added in Cisco Unified Communications Manager 7.1(5) and is available in 8.5(1) or later versions.

Note Upgrade from an unrestricted version to a restricted version is not supported.

Currently, the administrator is unable to create a new role with security groups and roles - ‘Standard CTI Secure Connection’ and ‘Standard CTI Allow Reception of SRTP Key Material’ as these roles are not available in unrestricted Cisco Unified Communications Manager.

In case of an upgrade from non-secure restricted Cisco Unified Communications Manager to unrestricted Cisco Unified Communications Manager, all the security features are disabled and standard CTI secure roles associated with the end user are removed. But, the custom administrative roles created with CTI secure privileges are not disabled in the Cisco Unified Communications Manager database.

In such cases, the application connects to the unrestricted Cisco Unified Communications Manager as a non-secure application as the CTIManager filters out the information about CTI secure roles.

Upgrading from a secure restricted Cisco Unified Communications Manager to an unrestricted Cisco Unified Communications Manager is not supported. To do so, you should first set the security mode of the secure restricted Cisco Unified Communications Manager to non-secure and then upgrade to unrestricted Cisco Unified Communications Manager.

Also, after an upgrade, the secure JTAPI application will not be able to connect to upgraded Cisco Unified Communications Manager version. To achieve this, the application must delete the existing certificates and disable secure connections.

If the application tries to register to the CTI ports or route points as secure phones in unrestricted Cisco Unified Communications Manager, the request fails and JTAPI throws CiscoRegistrationExceptionImpl with error code as CiscoJtapiException.CTIERR_USER_NOT_AUTH_FOR_SECURITY. However, in some scenarios the registration request may pass but is followed by CiscoTermRegistrationFailedEv with a new errorCode CTI_SECURITY_NOT_ALLOWED.

Interface Changes

See CiscoTermRegistrationFailedEv, on page 642

Message Sequences

See Unrestricted Unified CM, on page 1535

Backward Compatibility

This feature is backward compatible.

URI Dialing

Cisco Unified JTAPI provides CTI support for URI dialing using directory URIs. Cisco Unified JTAPI differentiates between directory numbers and directory URIs by the presence of the @ symbol. If an @ symbol is present, the address is a directory URI.

URI dialing is also supported for CTI Remote Devices. Remote destinations can be configured with directory URIs as the remote destination number.

Interface Changes

The following interfaces support directory URI addresses as the dialed digits or destination address:

- Call.connect(Terminal origterm, Address origaddr, java.lang.String dialedDigits)
- CallControlCall.consult(TerminalConnection tc, java.lang.String dialedDigits)
- CallControlConnection.redirect(java.lang.String destinationAddress)
- CallControlCall.transfer(java.lang.String address)
- CallControlForwarding(java.lang.String destAddress)

Message Sequence

No effect on the message sequence

Backward Compatibility

No backward incompatible changes

Version Format Change

In release 6.0, the Cisco Unified JTAPI version changed from a 4-digit format to a 5-digit format that is similar to the format used by Cisco Unified Communications Manager. The JTAPI version will remain similar to the Cisco Unified Communications Manager version. New interfaces let applications get the extended version number. See CiscoJtapiVersion, on page 255.

Backward Compatibility

This feature is backward compatible.

Verification Involving PSTN Reachability

The Verification Involving PSTN Reachability (VIPR) feature routes calls that are currently routed over PSTN, over the internet. For a normal VIPR call, JTAPI supports a VIPR call but no notification is sent to the application indicating that it is a VIPR call. Currently, VIPR calls are similar to Gateway or ICT calls.

When the quality of VIPR calls over an IP trunk drops below a certain threshold, the calls are automatically routed through PSTN. JTAPI supports this fallback but does not report this to applications. Whenever VIPR PSTN fallback happens, media is terminated and reestablished. Applications can view CiscoRTPInputStoppedEv, CiscoRTPOutputStoppedEv followed by CiscoRTPInputStartedEv and CiscoRTPOutputStartedEv indicating the same.

Interface Changes

There are no interface changes.

Message Sequences

See Verification Involving PSTN Reachability, on page 1577.

Backward Compatibility

This feature is backward compatible.

Video Capabilities and Multi-Media Information

In Cisco Unified Communications Manager 10.0(1), JTAPI is exposing video capabilities for supported terminals and calls. Video capabilities for near and far-end terminals include whether they are video-enabled, inter-operability with TelePresence, and the number of screens. Video attributes for calls will also be available to JTAPI applications which would include IP/port address, codec, and other information. Using the provided video terminal and call information, JTAPI applications will be able to better handle calls like routing incoming video-capable calls to agents with video-enabled terminals.

Exposing Multimedia Capability on CiscoTerminal

Cisco JTAPI provides a new API, getCiscoMultiMediaCapabilityInfo() on CiscoTerminal to expose the multimedia capabilities of the terminal.

The Video Capabilities and Multi-Media Information application can determine:

- the video capability (either video disabled or video enabled) of the device,
- the number of screens on a SIP device (only), and
- if the device supports interoperability with telepresence devices.

These capabilities are exposed on a new interface CiscoMultiMediaCapabilityInfo, which will have the following APIs to expose these capabilities:

- getVideoCapability(),
- getTelepresenceInfo(), and
- getScreenCount().

Exposing Changes in Multimedia Capability Via a New Provider Event

Any change in video capability of the terminal will be notified to the application by a new JTAPI event (CiscoProvTerminalMultiMediaCapabilityChangedEv). Video capability can be changed only from the Admin Device Configuration pages. Plugging in or out a Cisco Camera does not affect the video capability status, hence the new event is not triggered in this case. This event is a JTAPI provider event, and will be delivered only if the application has added provider observers. The terminal has to be in the registered state as a pre-condition for receiving this event.

Note A change in Multimedia Capability through CiscoProvTerminalMultiMediaCapabilityChangedEv will not be delivered to applications when the video capability of an SCCP Phone changes. In this case, the terminal will unregister and register back; therefore the application needs to update the video capability after the terminal is registered. See Scenario Three, on page 1537.

Exposing Multimedia Capability on a CiscoCall

An application can detect if the far-end Party for an incoming call is video capable prior to media setup. Consider a scenario where A calls B, the multimedia capabilities of the calling and called party will be exposed on the CiscoCall on terminal B after the call is offered to terminal B. The Cisco JTAPI provides the getCallingTerminalMultiMediaCapabilityInfo () and getCalledTerminalMultiMediaCapabilityInfo() APIs on the CiscoCall to expose the multimedia capabilities of the calling and called party in a call.

The same APIs can be used to determine the multimedia capabilities for an outgoing call, but note that the video capability will be known only after the call is answered. Consider a scenario where A calls B, B answers the call, the multimedia capabilities of the calling and called party will be exposed on the CiscoCall on terminal A after the call is answered by terminal B. The APIs getCallingTerminalMultiMediaCapabilityInfo() and getCalledTerminalMultiMediaCapabilityInfo() return CiscoMultiMediaCapabilityInfo.

Exposing Multimedia Streams Information on CiscoTerminal

The new JTAPI terminal event CiscoMultiMediaStreamsInfoEv will be delivered to a terminal observer to indicate multimedia streams information of a call. The multimedia streams information is exposed on the interface CiscoMultiMediaProperties, via the API getProperties() on CiscoMultiMediaStreamsInfoEv. The

Cisco JTAPI provides the multimedia streams information of the terminal after a call is connected. A MultiMedia Stream may include a video stream, a presentation stream, or both.

A video capable device is a device that can do any of the following:

- receive video (Video capability enabled in Admin Device Configuration pages and Cisco Camera not plugged in)
- send video (Video capability enabled in Admin Device Configuration pages and Cisco Camera plugged in)
- both send and receive video (Video capability enabled on Admin Device Configuration pages and Cisco Camera plugged in)

The following table describes the video capabilities that is provided by Cisco JTAPI for currently supported devices.

Phone Model	Protocol	Support Initial Device Multimedia Capability on CiscoTerminal	Supports Multimedia Capabilities on CiscoCall	Supports Multimedia Streams Information	Dynamic Video Capability Change
8945	SCCP	Yes	Yes	No	Yes
8945	SIP	Yes	Yes	Yes	Yes
9951/9971	SIP	Yes	Yes	Yes	Yes
EX60/90	SIP	Yes	Yes	Yes	N/A
CTIPort	SCCP	N/A	Yes	No	N/A
CTIRoutePoint	SCCP	N/A	Yes	No	N/A
CTS 500-32	SIP	Yes	Yes	Yes	N/A
Jabber (CSF/softphone mode)	SIP	Yes	Yes	Yes	N/A

Supported Features (Within the Same Cluster)

JTAPI will provide video capability information for same cluster calls involved in the following features:

- Originating Call and Consult Call
- Redirect
- Call Forward
- Hold and Resume
- Hunt List
- Transfer

- Super Provider
- Extension Mobility

Supported Features (Across Clusters)

JTAPI will provide video capability information for across-cluster calls involved in the following features:

- Originating Call and Consult Call
- Redirect
- Call Forward
- Hold and Resume
- Hunt List
- Super Provider
- Extension Mobility

Limitations

The following are the limitations of the Video Capabilities and Multi-Media Information feature:

- Outgoing call - Applications observing only calling party will have calling and called party multimedia capabilities as UNKNOWN until the called party answers the call. Refer to Scenario Eleven, on page 1547.
- Shared Line - Incoming call - calling and called party multimedia capabilities only if at least one of the terminal connections on the cisco call is not in passive state. Refer to Scenario Nine, on page 1543.
- Shared Line - Incoming Call - Called party multimedia capabilities will not have correct multimedia capabilities when more than one terminal connection is in ringing state. Refer to Scenario Ten, on page 1545.
- MultiMedia Streams Information - Cisco JTAPI will not deliver CiscoMultiMediaStreamsInfoEv on a CiscoTerminal which is a SCCP phone.
- Incoming Call - If an outbound call is initiated over SIP Trunk configured with Early Offer then the called party will just respond back with the capabilities it was offered during the initial offer and not its complete capabilities. Refer to Scenario Fifteen, on page 1560.
- Change in called party - In scenarios like Shared Lines or redirect, where the called party changes, the application will be notified of the new called party capability only if they configure the called party with unique display names.
- HuntList - Cisco JTAPI will not deliver correct multimedia capabilities for calls involving huntlist in broadcast mode.

Interface Changes

See the following sections for interface changes:

- CiscoCall, on page 326
- CiscoMasterKeyIndicator, on page 438
- CiscoMultiMediaCapabilityInfo, on page 462
- CiscoMultiMediaConnectionMode, on page 464
- CiscoMultiMediaEncryptionKeyInfo, on page 464
- CiscoMultiMediaProperties, on page 465
- CiscoMultiMediaStreamsInfoEv, on page 466
- CiscoMultiMediaType, on page 467
- CiscoProvTerminalMultiMediaCapabilityChangedEv, on page 483
- CiscoRTPPayload, on page 571
- CiscoRTPProperties, on page 572
- CiscoTermEvFilter, on page 608
- CiscoTerminal, on page 611

Message Sequences

See Video Capabilities and Multi-Media Information, on page 1536.

Backward Compatibility

This feature is backward compatible.

Video On Hold Support

In Cisco Unified Communications Manager Release 10.01, existing CiscoTerminalConnection.hold() API is enhanced to take an additional parameter - contentID. This enhancement was designed/developed for the Remote Expert solution. This newly added contentID is a pass through from application (JTAPI) to CCM. JTAPI will not process or manipulate this value. The contentID will reference a VoH stream to be played when the call is placed on hold.

The VoH files are housed externally on a media sense server. To have video on hold capability, the video on hold server must be configured in CCM Admin. This server coincides to the media sense server which houses all the VoH files.

Backward Compatibility

This feature is backward compatible and existing applications will not be affected by this enhancement.

Voice MailBox Support

This feature exposes voice mailbox numbers, which let Cisco Unified Communications Manager JTAPI applications forward calls from a directory number to the correct voice mailbox.

The Cisco Unified Communications Manager Administrator can associate a voicemail profile for each directory number. When the voicemail option is enabled for any forward setting, and if the corresponding forward is enabled, the call rolls down to the voicemail pilot number that is associated with the voicemail profile.

The voicemail profile contains voicemail pilot number and voice mailbox mask fields. Voice mailbox mask specifies the mask that is used to format the voice mailbox number for auto-registered phones. When forwarding a call to a voice messaging system from a directory line on an auto-registered phone, Cisco Unified Communications Manager applies this mask to the number that is configured in the Voice Mail Box field for that directory line.

For example, if you specify a mask of 972813XXXX, the voice mailbox number for directory number 7253 becomes 9728137253. If you do not enter a mask, the voice mailbox number matches the directory number (7253 in this example).

Cisco Unified Communications Manager JTAPI Support

To support this feature, Cisco Unified Communications Manager JTAPI exposes voice mailbox numbers for called party, lastRedirected party and originalCalled party. These voice mailbox fields are exposed on CiscoPartyInfo, which is exposed on CiscoCall object. If voicemail is not configured for a party, then Cisco Unified Communications Manager JTAPI will return empty Strings for voice mailbox fields.

In prior releases Cisco Unified Communications Manager JTAPI did not expose voice mailbox fields to applications, so Cisco Unified Communications Manager JTAPI voice mailbox applications could not determine whether a voice mailbox mask was configured for a voicemail profile, which could result in a voice mailbox number that differs from the directory number.

Performance and Scalability

This feature does not increase the traffic from the Cisco Unified Communications Manager JTAPI layer to the application layer. However, small performance impact could occur because of additional fields that are passed over the network.

XSI Object Pass Through

Applications can pass XML objects through JTAPI and CTI interfaces to the phone. The XML object can contain display updates, softkey update/enable/disable, and other types of updates on the phone that are available through IP phone services features. This allows applications to access IP phone service capabilities through JTAPI and CTI interfaces without maintaining independent connections to the phones.

CiscoTerminal Method

Applications can send an XSI object in the byte format to the Cisco Unified IPPhone through the CiscoTerminal interface method. The system limits the payload to 2000 bytes of data with this interface.

CiscoTerminal must be in the <CODE>CiscoTerminal.REGISTERED</CODE> state; its provider must be in the <CODE>Provider.IN_SERVICE</CODE> state. Successful response indicates that the data that was pushed has arrived at the phone; however, the application cannot receive any XML, including the CiscoIPPhoneResponse object from the push, back from the phone. If the application request is not successful, a PlatformException is thrown. Any request with more than 2000 bytes of data is rejected.

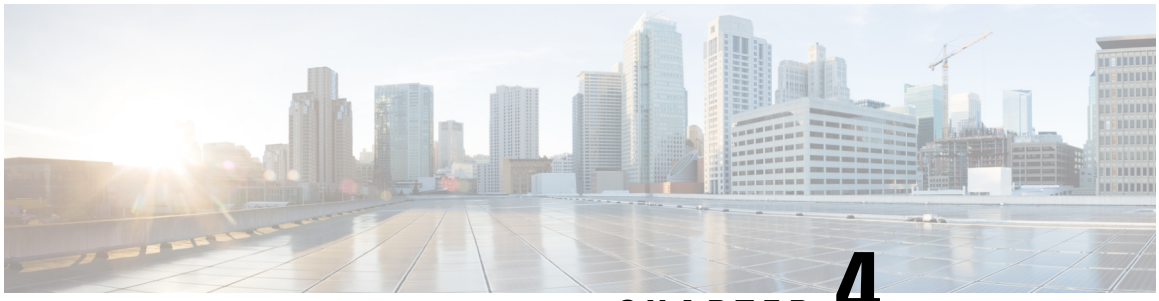
public String sendData (String terminalData) throws InvalidStateException, MethodNotSupportedException;

Before the application can make use of this feature, it must add TerminalObserver on the terminal.

Authentication and Mechanism

Sending an HTTP POST request to the phone web server, which requires the phone IP address, performs an object push. The web server parses the request, authorizes the request through the HTTP that is returned to the Cisco Unified Communications Manager, executes the request, and returns an XML response that indicates the success or failure of the request to the application.

With XSI, the IP phone services object gets sent directly to the phone by the Skinny Client Control Protocol (SCCP). The phone does not authenticate the request, because the JTAPI client is trusted and does not require the phone IP address. For more information on actual XML contents, refer to the Cisco IPPhone Services Application Development Notes.



CHAPTER 4

Cisco Unified JTAPI Installation

This chapter describes how to install and configure the Cisco Unified JavaTelephonyAPI (JTAPI) client software for Cisco Unified Communications Manager.

Overview

The Cisco Java Telephony API (JTAPI) implementation comprises Java classes that reside on all client machines that run JTAPI applications. Installation of the Cisco Unified JTAPI client must take place before these applications can function correctly. Make sure that the Cisco Unified JTAPI classes are installed wherever JTAPI applications run, whether on Cisco Unified Communications Manager Administration, a separate machine, or both.

Starting from **Release 11.5(1)SU9**, **Release 12.5(1)**, and any subsequent SU or ES releases in this release train, Cisco JTAPI Client for Linux, and Windows will be available as the zip file (.zip) which includes the JTAPI packages for Linux (32 and 64 bit) or Windows (32 and 64 bit), documentation, and sample codes. You can download the zip files (CiscoJTAPIWindows.zip or CiscoJTAPILinux.zip), by clicking the **Download** link corresponding to the Cisco JTAPI Client for Linux (32 and 64 bit) or Cisco JTAPI Client for Windows (32 and 64 bit) available in the Cisco Unified CM Administration interface, **Find and List Plugins** window (**Application** > **Plugins**).

The JTAPI Installer provides a unified installation/uninstallation process for the JTAPI client for Linux and Windows, as listed in the following table. For Cisco Unified Communications Manager 8.6(1) and later, Cisco JTAPI is also supported on 64-bit platforms. For Linux versions, the installer generates a binary file (.bin), and for the Windows version, it generates an executable file (.exe).

Supported JVM Versions for Cisco Unified Communications Manager Administration

For a detailed breakdown of supported JVM versions for this release of Cisco Unified Communications Manager, see https://developer.cisco.com/site/jtapi/documents/cisco-unified-jtapi-supported-jvm-versions/.

Note If you have upgraded from Cisco Unified Communications Manager Administration 4.x to 5.0 or later, you must upgrade the JTAPI client software on any application server or client workstation on which JTAPI applications are installed. If you do not upgrade the JTAPI client, your application fails to initialize.

Upgraded JTAPI client software does not work with previous releases of Cisco Unified Communications Manager.

Required Software

Cisco JTAPI requires the following software:

- Cisco Unified Communications Manager
- Supported Operating System Platform

Supported Platforms

For a detailed breakdown of supported Windows, Linux, and VMware platforms for Cisco Unified JTAPI, see https://developer.cisco.com/site/jtapi/documents/cisco-unified-jtapi-supported-jvm-versions/.

For additional information on virtualization within a Unified Communications environment, see http://docwiki.cisco.com/wiki/Virtualization_for_Cisco_Unified_Communications_Manager_(CUCM).

Installing the Cisco Unified JTAPI Software

Installation Procedures

The following sections describe the installation procedures for the Linux and Windows platforms.

From Release 11.5(1)SU9, the following installers are replaced with .zip files (CiscoJTAPIWindows.zip and CiscoJTAPILinux.zip).

- CiscoJTAPIClient-linux.bin
- CiscoJTAPIClient.exe
- CiscoJTAPIx64-Windows.exe
- CiscoJTAPIx64-Linux.bin

The following table lists the default JTAPI zip directory details. Classpath must be set accordingly to refer to the new jars.

Name/ Type of Client	JTAPI Libraries	Sample applications, documentation, and JTPrefs	CLASSPATH – 14SU2	CLASSPATH – Till 14SU2	LD_LIBRARY_PATH
CiscoJTAPIWindows.zip	{Unzip Location}\ CiscoJTAPIx32\lib	{Unzip Location}\ CiscoJTAPIx32	{Unzip Location}\CiscoJTAPIx32\lib\bc-fips.jar; {Unzip Location}\CiscoJTAPIx32\lib\bcpkix-fips.jar; {Unzip Location}\CiscoJTAPIx32\lib\bctls-fips.jar; {UnzipLocation}\CiscoJTAPIx32\lib\jtapi.jar	{Unzip Location}\CiscoJTAPIx32\lib\cryptojcommon.jar; {Unzip Location}\CiscoJTAPIx32\lib\cryptojce.jar; {Unzip Location}\ CiscoJTAPIx32\lib\jcmFIPS.jar; {Unzip Location}\ CiscoJTAPIx32\lib\sslj.jar; {Unzip Location}\CiscoJTAPIx32\lib\jtapi.jar	Not Applicable
	{Unzip Location}\ CiscoJTAPIx64\lib	{Unzip Location}\ CiscoJTAPIx64	{Unzip Location}\CiscoJTAPIx64\lib\ bc-fips.jar; {Unzip Location}\CiscoJTAPIx64\lib\bcpkix-fips.jar; {Unzip Location}\CiscoJTAPIx64\lib\bctls-fips.jar; {Unzip Location}\CiscoJTAPIx64 \lib\jtapi.jar	{Unzip Location}\CiscoJTAPIx64\lib\cryptojcommon.jar; {Unzip Location}\ CiscoJTAPIx64\lib\cryptojce.jar; {Unzip Location}\ CiscoJTAPIx64\lib\jcmFIPS.jar; {Unzip Location}\ CiscoJTAPIx64\lib\sslj.jar; {Unzip Location}\ CiscoJTAPIx64\lib\jtapi.jar	Not Applicable

Name/ Type of Client	JTAPI Libraries	Sample applications, documentation, and JTPrefs	CLASSPATH – 14SU2	CLASSPATH – Till 14SU2	LD_LIBRARY_PATH
CiscoJTAPILinux.zip	{Unzip Location}\ CiscoJTAPIx32\lib	{Unzip Location}\ CiscoJTAPIx32	export CLASSPATH=$CLASSPATH: {Unzip Location}\CiscoJTAPIx32\lib\ bc-fips.jar; {Unzip Location}\CiscoJTAPIx32\lib\bcpkix-fips.jar; {Unzip Location}\CiscoJTAPIx32\lib\bctls-fips.jar; {Unzip Location}\CiscoJTAPIx32 \lib\jtapi.jar	export CLASSPATH=$CLASSPATH: {Unzip Location}/ CiscoJTAPIx32/lib/CiscoJCEProvider.jar: {Unzip Location}/ CiscoJTAPIx32/lib/libCiscoJCEJNI.so: {Unzip Location}/ CiscoJTAPIx32/lib/libssl.so: {Unzip Location}/ CiscoJTAPIx32/lib/libssl.so.1.0.1: {Unzip Location}/ CiscoJTAPIx32/lib/log4j-1.2.17.jar: {Unzip Location}/ CiscoJTAPIx32/lib/libciscosafec.so: {Unzip Location}/ CiscoJTAPIx32/lib/libciscosafec.so.3: {Unzip Location}/ CiscoJTAPIx32/lib/libciscosafec.so.3.0.1: {Unzip Location}/ CiscoJTAPIx32/lib/libcrypto.so: {Unzip Location}/ CiscoJTAPIx32/lib/libcrypto.so.1.0.1: {Unzip Location}/ CiscoJTAPIx32/lib/slf4j-api-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx32/lib/slf4j-log4j12-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx32/lib/slf4j-simple-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx32/lib/jtapi.jar: {Unzip Location}/ CiscoJTAPIx32/lib/bcpkix-jdk15on-154.jar: {Unzip Location}/ CiscoJTAPIx32/lib/bcprov-jdk15on-154.jar **Note** Starting from release **12.5(1)SU5**, **14SU1** and any subsequent SU or ES releases in this release train, the JTAPI Linux plugin will bundle bcpkix-jdk15on.jar and bcprov-jdk15on.jar instead of bcpkix-jdk15on-154.jar and bcprov-jdk15on-154.jar. Classpath must be set accordingly to refer to the new jars.	export LD_LIBRARY_PATH= $LD_LIBRARY_PATH: {Unzip Location}/ CiscoJTAPIx32/lib **Note** The LD_LIBRARY_PATH need not be set from Release 14SU2.

Name/ Type of Client	JTAPI Libraries	Sample applications, documentation, and JTPrefs	CLASSPATH – 14SU2	CLASSPATH – Till 14SU2	LD_LIBRARY_PATH
	{Unzip Location}\ CiscoJTAPIx64\lib	{Unzip Location}\ CiscoJTAPIx64	export CLASSPATH=$CLASSPATH: {Unzip Location}\CiscoJTAPIx64\lib\bc-fips.jar; {Unzip Location}\CiscoJTAPIx64\lib\bcpkix-fips.jar; {Unzip Location}\CiscoJTAPIx64\lib\bctls-fips.jar; {Unzip Location}\CiscoJTAPIx64 \lib\jtapi.jar	export CLASSPATH=$CLASSPATH: {Unzip Location}/ CiscoJTAPIx64/lib/CiscoJCEProvider.jar: {Unzip Location}/ CiscoJTAPIx64/lib/libCiscoJCEJNI.so: {Unzip Location}/ CiscoJTAPIx64/lib/libssl.so: {Unzip Location}/ CiscoJTAPIx64/lib/libssl.so.1.0.1: {Unzip Location}/ CiscoJTAPIx64/lib/log4j-1.2.17.jar: {Unzip Location}/ CiscoJTAPIx64/lib/libciscosafec.so: {Unzip Location}/ CiscoJTAPIx64/lib/libciscosafec.so.3: {Unzip Location}/ CiscoJTAPIx64/lib/libciscosafec.so.3.0.1: {Unzip Location}/ CiscoJTAPIx64/lib/libcrypto.so {Unzip Location}/ CiscoJTAPIx64/lib/libcrypto.so.1.0.1: {Unzip Location}/ CiscoJTAPIx64/lib/slf4j-api-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx64/lib/slf4j-log4j12-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx64/lib/slf4j-simple-1.7.24.jar: {Unzip Location}/ CiscoJTAPIx64/lib/jtapi.jar: {Unzip Location}/ CiscoJTAPIx64/lib/bcpkix-jdk15on-154.jar: {Unzip Location}/ CiscoJTAPIx64/lib/bcprov-jdk15on-154.jar **Note** Starting from release **12.5(1)SU5**, **14SU1** and any subsequent SU or ES releases in this release train, the JTAPI Linux plugin will bundle bcpkix-jdk15on.jar and bcprov-jdk15on.jar instead of bcpkix-jdk15on-154.jar and bcprov-jdk15on-154.jar. Classpath must be set accordingly to refer to the new jars.	export LD_LIBRARY_PATH= $LD_LIBRARY_PATH: {Unzip Location}/ CiscoJTAPIx64/lib

Linux Platforms

Cisco Unified JTAPI supports multiple languages for the installation and JTAPI Preferences user interface.

The Cisco Unified JTAPIInstaller installs the following items on the local disk drive:

- JTAPI java classes in $HOME/.jtapi/lib
- JTAPI Preferences in $HOME/.jtapi/bin
- JTAPI sample applications (makecall, jtrace) in $HOME/.jtapi/bin
- JTAPI documentation in $HOME/.jtapi/bin/doc

Applicable from 8.6 release:

For 64-bit JTAPI Installer on 64-bit OS:

- JTAPI java classes in $HOME/.jtapi64/lib
- JTAPI Preferences in $HOME/.jtapi64/bin
- JTAPI sample applications (makecall, jtrace) in $HOME/.jtapi64/bin
- JTAPI documentation in $HOME/.jtapi64/bin/doc

Perform the following steps to install the Cisco Unified JTAPI software on a Linux platform:

1. Log in to the computer where you want to install the Cisco Unified JTAPI client software.
2. Locate the appropriate ISMP/IA installer and launch it:

Applicable from 8.6 release::

For 64-bit JTAPI Installer on 64-bit OS,

- CiscoJTAPIx64-Linux.bin - for Linux OS

1. Log in to the computer where you want to install the Cisco Unified JTAPI client software.
2. Locate the appropriate ISMP/IA installer and launch it:
3. Follow the instructions that the Cisco Unified JTAPI Installer presents.

Applicable for 11.5(1)SU9 and any subsequent SU or ES releases in this release train and also 12.5(1) release onwards.

Perform the following steps to install the Cisco Unified JTAPI software on a Linux platform:

Before you begin

- If you are using Cisco JTAPI version earlier to release 11.5(1)SU9, then uninstall the earlier version.

Step 1 Log in to the computer where you want to install the Cisco Unified JTAPI client software.

Step 2 Click **Download** link to download the required JTAPI client from the Unified Communications Manager Administrative interface Plugins page (**Application** > **Plugins**). Save the zipped file on the CTI application server where JTAPI is used.

Step 3 Un-zip the downloaded folder to extract the files. The files include the JTAPI packages for Linux (32-bit and 64-bit), documentation, and sample code. Update the classpath variable. **Go to Step 6.**

Step 4 Alternatively, run install32.sh or install64.sh depending on the platform . Follow the instructions mentioned in the scripts to install and update classpath.

Step 5 After installation, go to the installed location.

Step 6 Run the jtprefs.bat file.
The **Cisco Unified Communications Manager Jtapi Preferences <version number> Release** dialog box is displayed.

Note JTPrefs is an application, which provides a user interface to configure the jtapi.ini parameters. JTprefs is used to create the jtapi.ini file if one does not exist and to configure or modify the trace settings. In Linux Machines after installation the session must be logged out and logged in again for the changes to take effect.

In Linux, for example, the default directory is unzipped folder\CiscoJTAPILinux\CiscoJTAPIx64\lib or unzipped folder\CiscoJTAPILinux\CiscoJTAPIx32\lib.

Note The installation software installs the Cisco Unified JTAPI software on the default drive.

In Linux, for example, the default directory is $HOME/.jtapi/lib(for 32 bit installer); $HOME/.jtapi64/lib(for 64 bit installers).

Verifying Linux Installation

To ensure that the JTAPI installation has been done properly, perform the following steps:

Step 1 Check that the .jtapiver.ini file is created in the $HOME directory.

Step 2 Check that the JTAPI Program files and documentation are present under the folder $HOME/.jtapi/bin.

Look for the makecall, jtrace, Locale_files, and doc folders.

Step 3 Check that the JTAPI Library is present under the folder $HOME/.jtapi/lib.

Look for the jtapi.jar, jtracing.jar, and updater.jar files.

Step 4 After ensuring that jtapi.jar is present in the classpath, run the following command from the command line prompt of $HOME/.jtapi/bin ./_jvm/bin/java:

```
com.cisco.services.jtprefs.jtprefsFrame
```

The JTAPI Preferences dialog box appears.

Note In the absence of the JTPrefs application, you can generate the jtapi.ini file by entering:

```
< jview | java > CiscoJtapiVersion -parms
```

This command generates a jtapi.ini file in the current directory.

For Cisco Unified Communications Manager 8.6(1) and later, for 64 bit installer on a 64 bit OS, the default install directory is $HOME/.jtapi64/.

After installation, CLASSPATH is updated with the location of jtapi.jar. For linux a file .jtapiver.ini is updated with the install location in user home directory. For classpath changes to take effect, you need to log off and login again.

During installation, you can choose a different folder than $HOME to install JTAPI. In this case, the system creates a folder called .jtapi within the specified folder and creates the bin and lib folders within that folder for copying the corresponding files. For example, if you choose the folder name /home/jtapiuser, the folder structure would be

/home/jtapiuser/.jtapi/bin (for 32 bit installers)—Contains the makecall, jtrace, Locale_files, and doc folders.

or

/home/jtapiuser/.jtapi64/bin (for 64 bit installers)

/home/jtapiuser/.jtapi/lib(for 32 bit installers)—Contains the jtapi.jar, jtracing.jar, and updater.jar files

or

/home/jtapiuser/.jtapi64/lib (for 64 bit installers)

In this case, run the command at Step4 from the /home/jtapiuser/.jtapi/bin folder (for 32 bit installer) or /home/jtapiuser/.jtapi64/bin folder (for 64 bit installer).

Windows Platforms

Cisco Unified JTAPI supports multiple languages for the installation and JTAPI Preferences user interface.

The Cisco Unified JTAPI Installer installs the following items on the local disk drive:

Applicable from 8.6 release:

- JTAPI Java classes in %SystemRoot%\java\lib
- JTAPI Preferences in Program Files\JTAPITools
- JTAPI sample applications (makecall, jtrace) in Program Files\JTAPITools
- JTAPI documentation in Program Files\JTAPITools\doc

For 64-bit JTAPI Installer on 64-bit OS,

- JTAPI Java Preferences in Program Files\Cisco\JTAPI64Tools
- JTAPI Java classes in Program Files\Cisco\JTAPI64Tools\lib
- JTAPI sample applications (makecall, jtrace) in Program Files\Cisco\JTAPI64Tools
- JTAPI documentation in Program Files\Cisco\JTAPI64Tools\doc

Post installation, CLASSPATH is updated with the location of jtapi.jar. For windows, registry is updated, [HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems, Inc.\JTAPI\Client\Tools

HKEY_LOCAL_MACHINE\SOFTWARE\Cisco Systems, Inc.\JTAPI\Client\Tools\Lib]

To install the Cisco Unified JTAPI software on a Windows platform, perform the following steps:

1. Log in to the computer where you want to install the Cisco Unified JTAPI client software.
2. Close all Windows programs.
3. Locate the Cisco Unified JTAPI installer (CiscoJTAPIClient.exe) and launch it.
4. Follow the installer instructions.

Applicable for 11.5(1)SU9 and any subsequent SU or ES releases in this release train and also 12.5(1) release onwards.

Perform the following steps to install the Cisco Unified JTAPI software on a Windows platform:

Before you begin

- If you are using Cisco JTAPI version earlier to release 11.5(1)SU9, then uninstall the earlier version.

Step 1 Log in to the computer where you want to install the Cisco Unified JTAPI client software.

Step 2 Click Download link to download the required JTAPI client from the Unified Communications Manager Administrative interface Plugins page (**Application** > **Plugins**). Save the zipped file on the CTI application server where JTAPI is used..

Step 3 Un-zip the downloaded folder to extract the files. The files include the JTAPI packages for Windows (32-bit and 64-bit), documentation, and sample code. Update the classpath variable. Go to Step 6.

Step 4 Alternatively, run install32.bat or install64.bat depending on the platform . Follow the instructions mentioned in the scripts to install and update classpath.

Step 5 After installation, go to the installed location.

Step 6 Run the jtprefs.bat file.
The **Cisco Unified Communications Manager Jtapi Preferences <version number> Release** dialog box is displayed.

Note JTPrefs is an application, which provides a user interface to configure the jtapi.ini parameters. JTprefs is used to create the jtapi.ini file if one does not exist and to configure or modify the trace settings.

In Windows, for example, the default directory is unzipped folder\CiscoJTAPIWindows\CiscoJTAPIx64\lib or unzipped folder\CiscoJTAPIWindows\CiscoJTAPIx32\lib.

Verifying Windows Installation

To verify the JTAPI Windows installation, you can use the makecall application that allows you to place a call via JTAPI. Perform the following steps to use the makecall application.

Step 1 From the Windows command line, navigate to the directory where you installed Cisco Unified JTAPI Tools. By default, this directory is C:\ProgramFiles\JTAPITools (for 32 bit installers) and C:\ProgramFiles\JTAPI64Tools (for 64 bit installers).

Step 2 Execute the following command:

java CiscoJtapiVersion

Step 3 Execute the following command:

java makecall <server name> <login> <password> **1000** <phone1> <phone2>

Note The server name variable specifies the hostname or IP address of the Cisco Unified Communications Manager (for example, 192.168.1.100 or Subscriber2).

The phone1 and phone2 variables designate directory numbers of IP phones or virtual phones that the user controls according to the user configuration. Refer to the chapter ‘Directory Number Configuration’ in Cisco Unified Communications Manager Administration Guide for details.

For the login and password variables, use the user ID and password that you configured in the Cisco Unified Communications Manager User Configuration window.

Linux and Windows Installation

Reinstall or Upgrade or Downgrade

This feature provides a uniform install and uninstall procedure for the Cisco Unified JTAPI Client on Linux and Windows platforms.

Starting from Release 11.5(1)SU9, Release 12.5(1), and any subsequent SU or ES releases, the Linux and Windows versions generate the zip file (.zip) which includes the JTAPI packages for Linux (32 and 64 bit) or Windows (32 and 64 bit), documentation, and sample codes.

You can download the zip files (CiscoJTAPIWindows.zip or CiscoJTAPILinux.zip), by clicking the **Download** link available in the Cisco Unified CM Administration interface, **Find and List Plugins** window (**Application** > **Plugins**).

From the Unified Communications Manager release 12.5(1) to the 14SU2 plugin URL, you can download the CiscoJ Libraries of Linux (32 and 64 bit):

- https://<*IP address*>/plugins/lib_ciscoj_x32.zip
- https://<*IP address*>/plugins/lib_ciscoj_x64.zip

To reinstall or upgrade, replace the existing files with the newly extracted files downloaded from the Cisco Unified CM Administration interface. For more details, see the "Installation Procedure" chapter.

To reinstall or downgrade or install the JTAPI version earlier to release 12.5(1), excluding 11.5(1)SU9 ESs and SU releases, refer to the "Cisco Unified JTAPI Installation chapter.

To determine the JTAPI version for both Windows and Linux platforms, run the following command:

- java CiscoJtapiVersion

Using Cisco Unified CM JTAPI

The following section describes the program group and program elements created by the installation of Cisco JTAPI.

Program Group and Program Elements

After the installation of Cisco JTAPI, a program group called CiscoJTAPI is created which contains the following elements:

- Cisco Unified Communications Manager JTAPI Javadocs — Opens the Javadocs reference guide for Cisco JTAPI.
- Cisco Unified Communications Manager JTAPI Preferences — Launches the JTAPI Preferences application.

- ReadMe — Launches the readme.htm file in the default web browser.
- Updater Javadocs — Opens the Javadocs Updater package that is bundled with Cisco JTAPI.

Cisco Unified JTAPI Configuration Settings

You can use the Cisco Unified JTAPI Preferences application to configure trace levels and trace destinations as well as several other system parameters.

Note When using Windows 7 and Windows 2008 Server, you must run the JTAPI Preferences application in Administrative Mode when the User Access Control (UAC) service is running. If the UAC service is disabled, the JTAPI Preference application can run without administrative privileges.

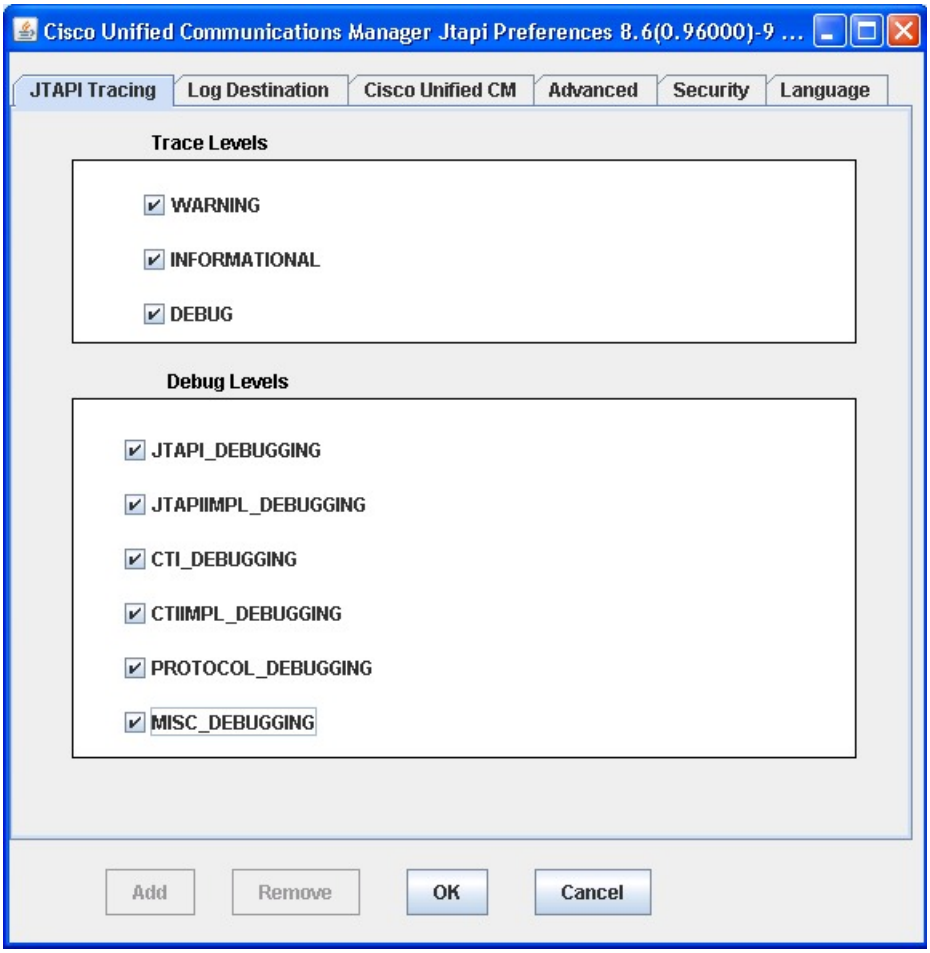
This section, which describes how to use the Cisco Unified JTAPI Preferences application, includes the following topics:

- JTAPI Tracing Tab, on page 223
- Log Destination Tab, on page 225
- Cisco Unified CM Tab, on page 227
- Advanced Tab, on page 228
- Security Tab, on page 231
- Language Tab, on page 233

JTAPI Tracing Tab

The JTAPI Tracing tab lets you change trace settings for the JTAPI layer. The following figure illustrates the JTAPI Tracing tab of the Cisco Unified JTAPI Preferences application. The window title shows the JTAPI version number.

Figure 14: JTAPI Tracing Tab



The JTAPI Tracing tab lets you enable or disable JTAPI trace levels as listed in the following table.

Table 10: JTAPI Trace Levels

Jtapi.ini fields	Description
Trace Levels	
WARNING	Low-level warning events
INFORMATIONAL	Status events
DEBUG	Highest level debugging events
Debug Levels	
JTAPI_DEBUGGING	JTAPI methods and events trace
JTAPIIMPL_DEBUGGING	Internal JTAPI implementation trace

Jtapi.ini fields	Description
CTI_DEBUGGING	Trace Cisco Unified Communications Manager events that are sent to JTAPI
CTIIMPL_DEBUGGING	Internal CTICLIENT implementation trace
PROTOCOL_DEBUGGING	Full CTI protocol decoding
MISC_DEBUGGING	Miscellaneous low-level debug trace

Log Destination Tab

The Log Destination tab allows you to configure how JTAPI creates traces and where they are stored. The following figure illustrates the Log Destination tab of the Cisco Unified JTAPI preferences application. The following table contains descriptions of the log destination fields.

Figure 15: Log Destination Tab

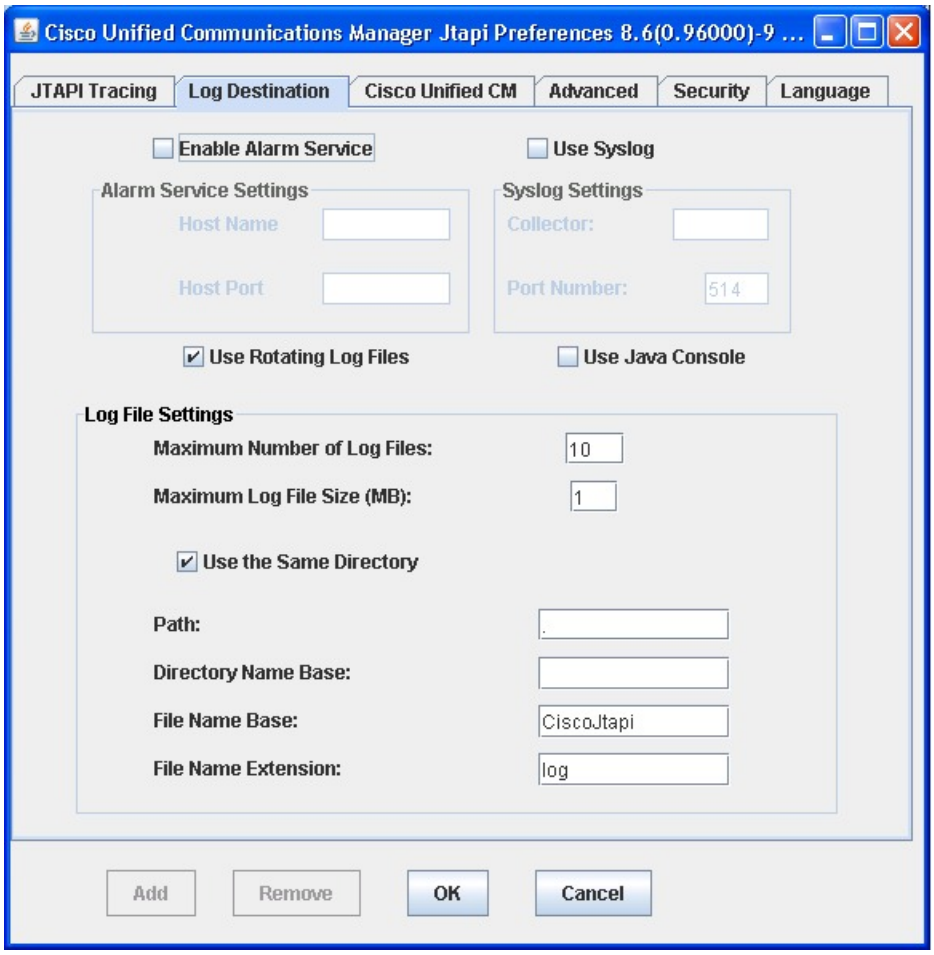


Table 11: Log Destination Fields

Field name	Default	Min	Max	Description
Enable Alarm Service(UseAlarmService)	0	Not Applicable (NA)	NA	When this option is enabled, JTAPI alarms go to an alarm service that is running on the specified machine. You must specify the host name and port number when you enable this option.
Use Syslog (UseSyslog)	FALSE	NA	NA	When this option is enabled, traces go to a UDP port as specified in the Collector and Port Number fields. Syslog collector service collects traces and directs them to the Cisco Operations Manager Suite server.
Alarm Service Settings				
Host Name		NA	NA	Use this field to specify the host name of the alarm service server.
Host Port		NA	NA	Use this field to specify the host port of the alarm service server.
Syslog Settings				
Collector	0	NA	NA	Use this field to specify the Syslog collector service that collects traces.
Port Number	514	NA	NA	Use this field to specify the UDP port of the collector.
Use Rotating Log Files (SyslogCollector)	FALSE	NA	NA	This field allows you to direct traces to a specific path and folder. No fewer than two log files and no more than 99 files can exist. Cisco Unified JTAPI rotates through the log files in numerical order, returning to the first log file after filling the last. Log files increase in size in 1-megabyte increments.
Use Java Console (UseSystemDotOut)	FALSE	NA	NA	When this option is enabled, tracing goes to the standard output or console (command) window.
Log File Settings				
Maximum Number of Log Files (NumTraceFiles)	10	2	1000	This setting lets you specify the maximum number of log files to be written.
Maximum Log File Size (TraceFileSize)	1048576	1048576	NP	This setting lets you specify the maximum size of log files to be written.

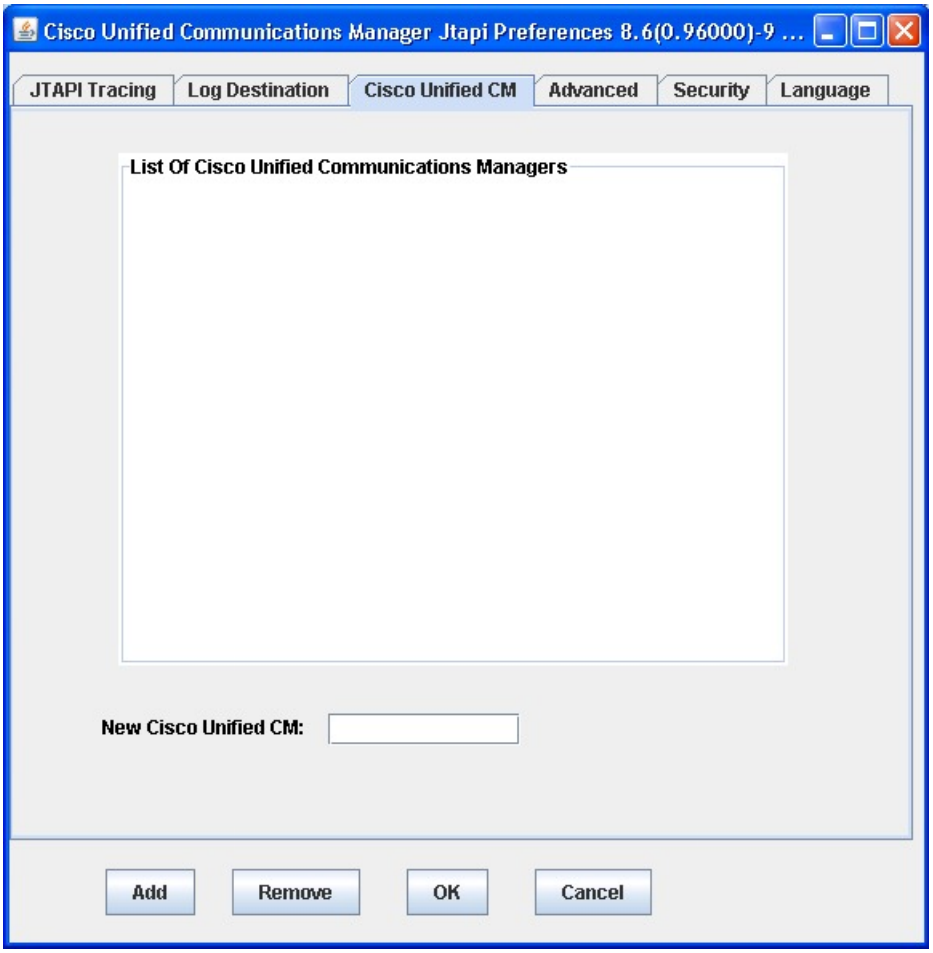
Field name	Default	Min	Max	Description
Use the Same Directory (UseSameDirectory)	1	NA	NA	This setting lets you specify whether the same folder name should be used for each instance of an application. When this option is enabled, JTAPI traces the log files to the same directory. In this case, successive instances of a JTAPI application will restart the log files, starting at index 01. When this option is disabled, each instance of the application, whether successive or simultaneous, will cause trace files to be placed in a new folder sequential to the last folder that was written. Cisco Unified JTAPI detects the last folder present in the trace path and automatically increments the numeric index.
Trace Path (TracePath)	.	NA	NA	This setting lets you specify the path name to which trace files are written. When the path is not specified, JTAPI defaults to the application path.
Directory Name Base (Directory)	.	NA	NA	This setting lets you specify a folder name where the trace files will be contained.
File Name Base (FileNameBase)	Cisco Jtapi	NA	NA	Use this value to create the trace file name.
File Name Extension (FileNameExtension)	log	NA	NA	This setting lets you specify a numerical index to append to the file base name indicates the order in which trace files are created. If you enter “jtapiTrace” in the File Name Base field and “log” in the File Name Extension field, the system names the trace files jtapiTrace01.log, jtapiTrace02.log, and so on. If the File Name Base and File Name Extension fields are left blank, JTAPI picks the trace files names as CiscoJtapi01.log, CiscoJtapi02.log, and so on.

Cisco Unified CM Tab

This tab allows you to define a list of IP addresses for Cisco Unified Communications Manager Subscribers where CTIManager is enabled. Applications can query JTAPI for this list and use it to find the IP addresses to connect to. You can define a maximum 10 IP addresses.

The following figure illustrates the Cisco Unified CM tab of the preferences application.

Figure 16: Cisco Unified CM Tab

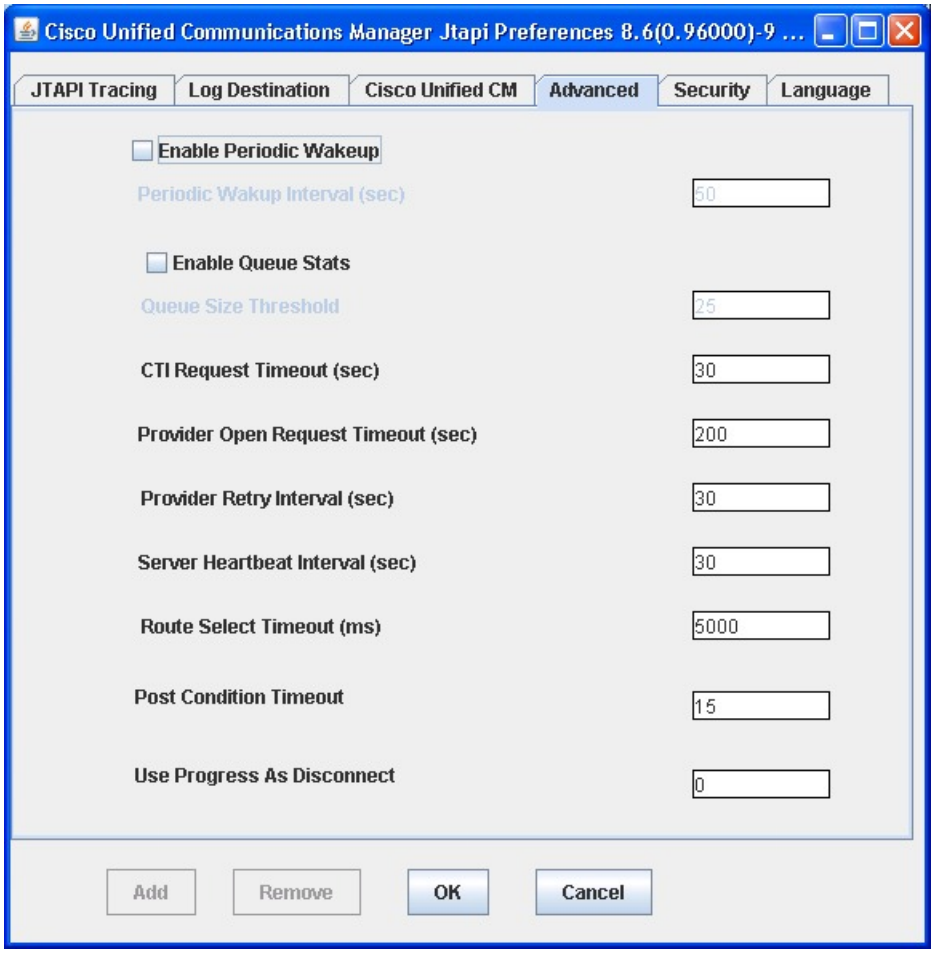


Advanced Tab

You can configure the parameters in the table in this section through the Advanced tab in the JTAPI Preferences application. These low-level parameters are used for troubleshooting and debugging purposes only.

The following figure illustrates the Advanced tab of the preferences application.

Figure 17: Advanced Tab



Note Cisco strongly recommends that you not modify the parameters in the following table unless the Cisco Technical Assistance Center (TAC) instructs you to do so.

Table 12: Advanced Configuration Fields

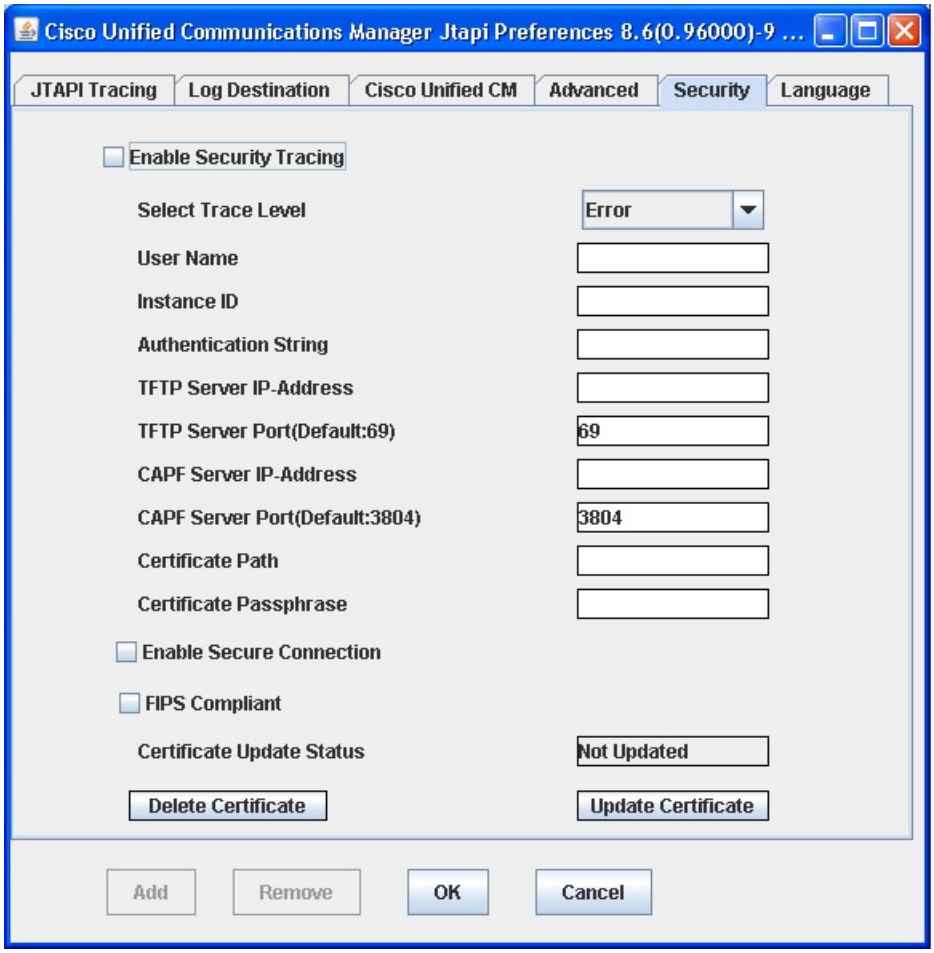
Field	Default	Min	Max	Description
Enable Periodic Wakeup(PeriodicWakeupEnabled)	FALSE	Not Applicable (NA)	NA	Enables (or disables) a heartbeat in the internal message queue that JTAPI uses. If JTAPI has not received a message in the time that is defined in PeriodicWakeupInterval, it causes the thread to wake up and creates a log event.

Field	Default	Min	Max	Description
Periodic Wakeup Interval(PeriodicWakeupInterval)	50	Not Present (NP)	NP	Allows you to define a period of inactivity in the JTAPI internal message thread (in seconds). If JTAPI has not received a message during this time, the thread wakes up and logs an event.
Enable Queue Stats(QueueStatsEnabled)	FALSE (disabled)	NA	NA	Causes JTAPI to log the max queue depth over the specified number of messages that are queued to JTAPI main event thread. For every x messages processed, JTAPI logs a DEBUGGING level trace that reports the maximum queue depth over that interval, where x represents the number of messages that are specified in Queue Size Threshold.
Queue Size Threshold(QueueSizeThreshold)	25	10	NP	Specifies the number of messages that define the interval over which JTAPI will report the maximum queue depth.
CTI Request Timeout(CtiRequestTimeout)	15	10	NP	Specifies the number of seconds that JTAPI will wait for a response from a CTI request.
Provider Open Request Timeout(ProviderOpenRequestTimeout)	200	10	NP	Specifies the number of seconds that JTAPI will wait for a response to a Provider Open Request.
Provider Retry Interval(ProviderRetryInterval)	30	5	NP	Specifies the number of seconds that JTAPI will retry opening connection to a Cisco Unified Communications Manager cluster in case of system failure.
Server Heartbeat Interval(DesiredServerHeartbeatInterval)	30	>0	NP	Specifies the interval at which the connection between JTAPI and the Cisco Unified Communications Manager cluster will get verified (in seconds). If JTAPI fails to receive heartbeats, it will establish a connection via the second CTIManager that is specified in the provider open request.
Route Select Timeout(RouteSelectTimeout)	5000	0	NP	Specifies the interval in milliseconds that JTAPI will wait for the application to respond to the Route event. If the application does not respond in this time, JTAPI ends the route and sends the corresponding RouteEnd event.
Post Condition Timeout	15	0	NP	Specifies the timeout.

Security Tab

The following figure illustrates the Security tab of the preferences application.

Figure 18: Security Tab



Administrators need to configure the User Name, Instance ID, Authorization Code, TFTP Server IP-Address, and CAPF Server IP-Address parameters through the JTAPI Preferences application before invoking the JTAPI API or JTAPI Preferences to download/install certificates on the application server.

You can use JTAPI Preferences to configure security profiles for one or more User Name/Instance ID pairs. If an application user has previously configured a security profile for a User Name/Instance ID pair, the security profile automatically populates when the user enters the User Name/Instance ID and clicks any of the other edit boxes.

Apart from the GUI that is provided through JTAPI Preferences, an application can also install a client certificate by calling the interface that is provided at CiscoJtapiProperties. When Interface UpdateCertificate is called, the JTAPI client connects to the TFTP server to download the CTL file and extract certificates to the given certificate path. It then connects to the CAPF server to download the client certificate and installs it into the given certificate path.

The jtapi.ini files store user security records in comma separated value (CSV) format. Semicolons separate individual records. An example of a users security record is as follows:

SecurityProperty = user, 123, 12345, 172.19.242.37, 3804, 172.19.242.37, 69, .\\, true, false;<next record>;
…

You can configure the following parameters on the Security tab:

Table 13: JTAPI Security Configuration Fields

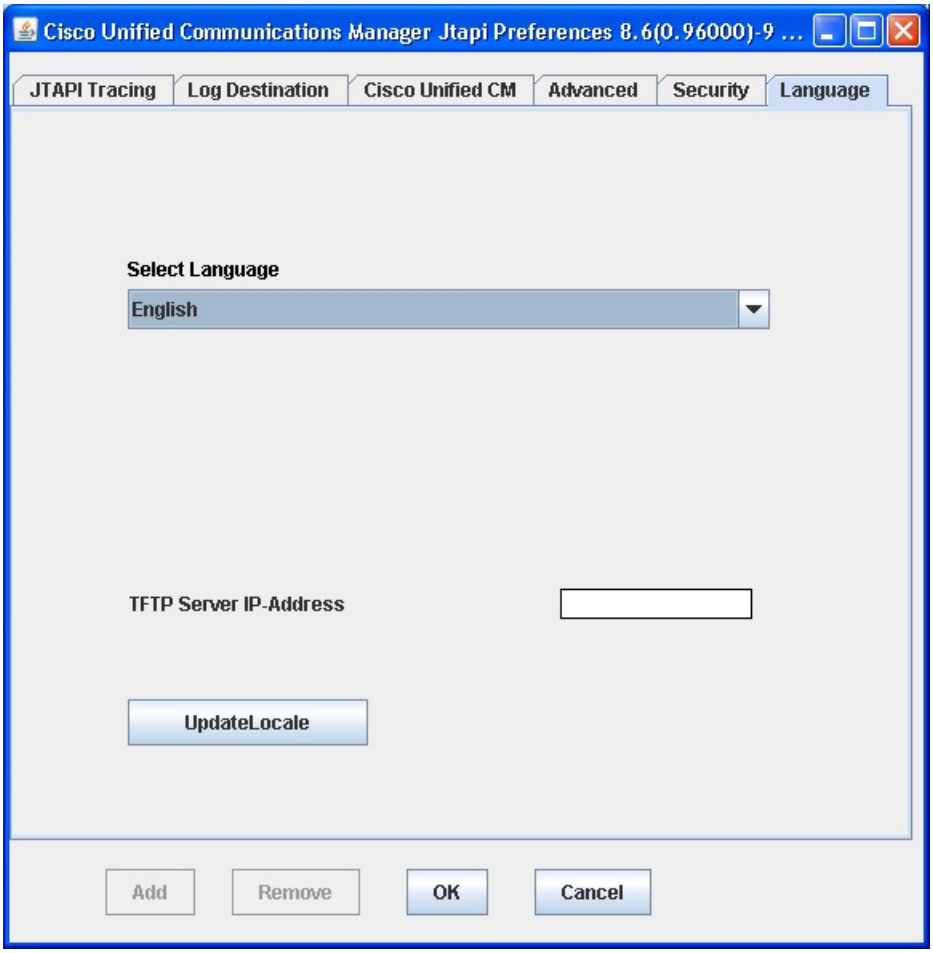
Field	Default	Min	Max	Description
Enable Security Tracing(SecurityTraceEnabled)	FALSE	Not Applicable (NA)	NA	You can enable (or disable) tracing for certificate install operations by checking this check box and choosing the desired trace level.
Select Trace Level(SecurityTraceLevel)	0	0	2	You can choose one of three different trace levels: • Error = 0 — Logs error events • Debug = 1 — Logs debugging events • Detailed = 2 — Logs all events
User Name(Username)	NA	NA	NA	If application users have previously configured a security profile for a User Name/Instance ID pair, that security profile automatically populates when the user enters the User Name/Instance ID and clicks any of the other edit boxes.
Instance ID(instanceID)	NA	NA	NA	This field specifies the application instance identifier. If an application is connecting to CTIManager with the same user, it needs to define an instanceID for each instance of the application to download the certificate Authorization String.
Authentication String(authcode)	NA	NA	NA	This field specifies a one-time string that is used to download a certificate.
TFTP Server IP Address	NA	NA	NA	This field specifies the IP address of the TFTP server (normally the Cisco Unified Communications Manager IP Address).
TFTP Server Port	69	Not Present (NP)	NP	The TFTP Server Port defaults to 69. Do not change this value unless the System Administrator advises you to do so.
CAPF Server IP Address	NA	NA	NA	This field specifies the IP address of the CAPF server in dotted decimal.

Field	Default	Min	Max	Description
CAPF Server Port	3804	NP	NP	The CAPF Server Port number defaults to 3804; however, you can also configure this number in the Cisco Unified Communications Manager Administration. Ensure that the value that is entered through the JTAPI Preferences matches the one that is configured in Cisco Unified Communications Manager Administration.
Certificate Path	JTAPI. jarlocation	NA	NA	This field specifies the path where the application wants server and client certificates to be installed. If this field is blank, the system installs certificates in the ClassPath of JTAPI.jar.
Enable Secure Connection	FALSE	NA	NA	Check this option to enable a secure TLS connection to Cisco Unified Communications Manager. If this option is not checked, JTAPI cannot make a nonsecure connection to CTI even if the certificate is updated/installed.
Certificate Update Status	NA	NA	NA	This field provides information on whether the certificate has been updated.
Delete Certificate	NA	NA	NA	This button deletes the existing certificate.
Update Certificate	NA	NA	NA	This button updates the existing certificate with the changed parameters.
FIPS Compliant	FALSE	NA	NA	Check this option to enable JTAPI to be FIPS compliant.

Language Tab

The following figure illustrates the Language tab of the preferences application.

Figure 19: Language Tab



The Language tab allows you to select one of the installed languages to view the configuration settings in that language.

Note You must install the language pack on the TFTP server before using this feature.

You can select the following languages:

Arabic	Brazilian Portuguese	Chinese Taiwan	Croatian	Czech
Danish	Dutch	English	Finnish	French
German	Greek	Hebrew	Hungarian	Italian
Japanese	Nederlands	Norwegian	Polish	Portuguese
Russian	Simplified Chinese	Slovak	Spanish	Swedish

Select a language, and the tabs display with text in that language.

Managing the Cisco Unified CM JTAPI

You can perform the following actions on all the Cisco JTAPI clients.

Reinstalling, Upgrading or Downgrading the Cisco JTAPI

Applicable from 11.5(1)SU9 and 12.5(1) ES and SU releases only.

Use the following procedure to reinstall or upgrade or downgrade the Cisco JTAPI client on all supported platforms from release 12.5(1) and later versions.

Before you begin

Note To reinstall or upgrade the JTAPI version earlier to release 12.5(1), refer to the "Cisco Unified JTAPI Installation".

Step 1 Delete the contents of the previous zip folder present in the system and clear the classpath variables . Alternatively, you can run the Uninstall Script present in the installed folder to delete the files and update classpath.

Step 2 Click **Download** link to download the required JTAPI client from the Unified Communications Manager Administrative interface Plugins page (**Application** > **Plugins**). Save the zipped file on the CTI application.

Step 3 Un-zip the downloaded folder to extract the files . Manually update the classpath variables. Additionally, you can run the Install Script present in the extracted folder to install Cisco JTAPI and update the classpath.

Note You can keep a copy of the jtapi.ini file present in [{Unzip Location}\lib\jtapi.ini] location and replace the same in the newly extracted Zip folder if you want to keep the previous Jtapi settings. It is applicable only for upgrading/downgrading/reinstallation.l

In Linux Machines, after uninstallation the session must be logged out and logged in again for the changes to take effect.

Uninstalling the Cisco JTAPI

Uninstalling the Cisco JTAPI

To remove the JTAPI version release 12.5(1), delete the folder (CiscoJTAPIWindows or CiscoJTAPILinux) and its extracted files from the system.

1. Delete the contents of the previous zip folder present in the system and clear the classpath variables.
2. You can also run the Uninstall Script present in the extracted folder to delete the files and update classpath. Run **uninstall32.bat** or **uninstall32.sh** for 32-bit machines and uninstall64.bat or uninstall64.sh for 64-bit machines.
3. Follow the instructions mentioned in the script.

To uninstall the JTAPI version earlier to release 12.5(1), refer to the "Cisco Unified JTAPI Installation" chapter of the *Cisco Unified JTAPI Developers Guide for Cisco Unified Communications Manager, Release 11.5(1)*, at https://www.cisco.com/c/en/us/support/unified-communications/unified-communications-manager-callmanager/products-programming-reference-guides-list.html.

Administering User Information for JTAPI Applications

The JTAPI application requires that users be given the privilege to control one or more devices. Follow the procedures for adding an application user and assigning devices to an application user in the "Application user setup" chapter of the *Cisco Unified Communications Manager Administration Guide* before using the JTAPI application. The list of devices that are assigned to the user represents the phones that the user needs to control from the application (for example, make calls and answer calls).

Fields in the jtapi.ini File

Applications that run in non-GUI based platforms, where the JTAPI Preferences application cannot be invoked, can write their own jtapi.ini file and place it along with jtapi.jar based on the values that are provided here. JTAPI will make use of these values.

Applications should ensure that they provide valid data as described in the following table. Applications are responsible for errors that are caused in JTAPI behavior due to improper jtapi.ini file values.

Table 14: Fields in jtapi.ini File

Jtapi.ini fields	Default	Min	Max	Description
INFORMATIONAL	0	Not Applicable (NA)	NA	This field specifies status events
DEBUG	0	NA	NA	This field specifies highest level debugging events
WARNING	0	NA	NA	This field specifies low-level warning events
JTAPI_DEBUGGING	0	NA	NA	This field specifies JTAPI methods and events trace
JTAPIIMPL_DEBUGGING	0	NA	NA	This field specifies internal JTAPI implementation trace
CTI_DEBUGGING	0	NA	NA	This field specifies trace Cisco Unified Communications Manager events that are sent to the JTAPI implementation
CTIIMPL_DEBUGGING	0	NA	NA	This field specifies internal CTICLIENT implementation trace
PROTOCOL_DEBUGGING	0	NA	NA	This field specifies full CTI protocol decoding

Jtapi.ini fields	Default	Min	Max	Description
MISC_DEBUGGING	0	NA	NA	This field specifies miscellaneous low-level debug trace
DesiredServerHeartbeatInterval	30	>0	Not Present (NP)	This field specifies how often, in seconds, the connection between JTAPI and the Cisco Unified Communications Manager cluster will be verified. If JTAPI fails to receive heartbeats, it will establish a connection via the second CTIManager that is specified in the provider open request.
TracePath	.	NA	NA	This field specifies the path name to which the trace files are written. When the path is not specified, JTAPI makes the application path as the default.
FileNameExtension	log	NA	NA	This field specifies a numerical index that is appended to the file base name to indicate the order in which the files are created. For example, if you enter jtapiTrace in the File Name Base field and log in the File Name Extension field, the trace files would rotate between jtapiTrace01.log, jtapiTrace02.log, and jtapiTrace10.log. If the File Name Base and File Name Extension fields are left blank, Cisco Unified JTAPI picks the trace files names as CiscoJtapi01.log, CiscoJtapi02.log, and so on.
SyslogCollector	FALSE	NA	NA	This field specifies you to direct the traces to a specific path and folder in the system. No fewer than two log files and no more than 99 files can exist. Cisco Unified JTAPI rotates through the log files in numerical order, and returns to the first log file after filling the last. Log files increase in size in 1-megabyte increments.
TraceFileSize	1048576	1048576	NP	This field allows you to specify the maximum size of log files to be written.
UseAlarmService	0	NA	NA	When this option is enabled, JTAPI alarms go to an alarm service that is running on the specified machine. You must specify the host name and port number if you enable this option.
ProviderOpenRequestTimeout	200	10	NP	This field specifies the time in seconds that JTAPI will wait for a response for the Provider Open Request. The default is 10 seconds.

Jtapi.ini fields	Default	Min	Max	Description
JtapiPostConditionTimeout	15	10	20	JTAPI has post conditions for events, and if the post condition is not met before a timeout, JTAPI will throw exceptions. Use this field to set the timeout value of such conditions.
ApplicationPriority	2	NA	NA	This field prioritizes multiple provider open requests. Currently, JTAPI only sends a default value.
SecurityTraceEnabled	FALSE	NA	NA	This field enables tracing for security-related messages. You can enable (or disable) tracing for certificate install operations by selecting this check box and selecting the desired trace level.
AlarmServicePort	1444	NP	NP	This field is used for sending alarms to a different server. Users can select the alarm server host name and port on which the service is running, and JTAPI will send the alarms to the specified server and port.
AlarmServiceHostname	null	NA	NA	This field displays the alarm server host name.
RouteSelectTimeout	5000	0	NP	This field specifies the time, in milliseconds, that JTAPI waits for the application to respond to the Route event. If the application does not respond in this time, JTAPI ends the route and sends the corresponding RouteEnd event.
ProviderRetryInterval	30	5	NP	This field specifies the time, in seconds, that JTAPI will retry opening a connection to the Cisco Unified Communications Manager cluster in case of system failure.
QueueStatsEnabled	FALSE	NA	NA	This field is used by JTAPI to log the max queue depth over the specified number of messages that are queued to JTAPI main event thread. In other words, for every x messages processed, JTAPI logs a DEBUGGING level trace that reports the maximum queue depth over that interval, where x represents the number of messages that are specified in Queue Size Threshold.
FileNameBase	CiscoJtapi	NA	NA	This field specifies a value to create the trace file name.

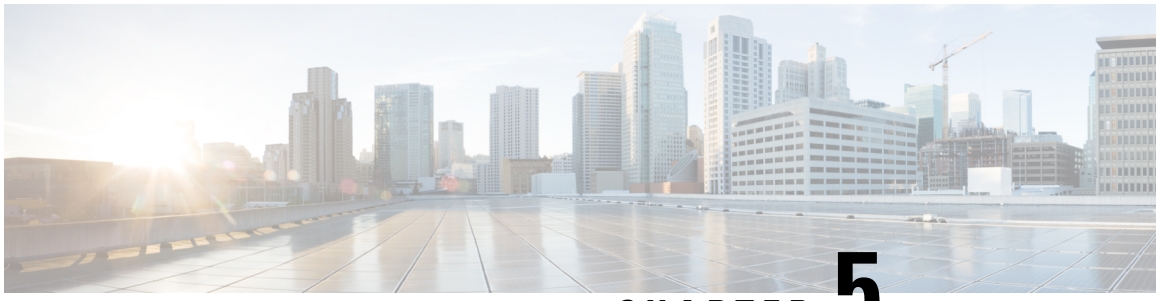
Jtapi.ini fields	Default	Min	Max	Description
PeriodicWakeupEnabled	FALSE	NA	NA	This field enables (or disables) a heartbeat in the internal message queue that JTAPI uses. If JTAPI has not received a message in the time that is defined in PeriodicWakeupInterval, it causes the thread to wake up and creates a log event.
JTAPINotificationPort	2789	1	NP	This field specifies the Port through which the JTAPI parameter changes are communicated to JTAPI applications during runtime.
PeriodicWakeupInterval	50	NP	NP	This field allows you to define a time of inactivity in the JTAPI internal message thread. If JTAPI does not received a message during this time, the thread wakes up and logs an event.
QueueSizeThreshold	25	10	NP	This field allows you to specify the number of messages that define the time over which JTAPI will report the maximum queue depth.
UseSystemDotOut	FALSE	NA	NA	This field is used to display traces on the console.
UseSameDirectory	1	NA	NA	This field allows you to specify whether the same folder name must be used for each instance of an application. When this option is enabled, JTAPI traces the log files to the same directory. In this case, successive instances of a JTAPI application will restart the log files, starting at index 01. When this option is disabled, each instance of the application, whether successive or simultaneous, will cause the trace files to be placed in a new folder sequential to the last folder that was written. Cisco Unified JTAPI detects the last folder in the trace path and automatically increments the numeric index.
NumTraceFiles	10	2	1000	This field allows you to specify the maximum number of log files to be written.

Jtapi.ini fields	Default	Min	Max	Description
UseSyslog	FALSE	NA	NA	This field, when enabled, allows the traces go to a UDP port as specified in the Collector and Port Number fields. Syslog collector service collects traces and directs them to the Cisco Operations Manager Suite server.
SecurityTraceLevel	0	0	2	This field specifies trace level for security messages 0 = Error, 1 = debug, 2 = detailed
UseTraceFile	TRUE	NA	NA	This field enables the writing of logs to logFile Trace Writer.
CMAssignedAppID	0	NA	NA	This field specifies the feature ID that is assigned to the application. Cisco Unified Communications Manager preassigns this ID.
CtiManagers	null	NA	NA	This field specifies the list of CTI Managers for which tracing needs to be collected.
Directory	.	NA	NA	This field allows you to specify a folder name where the trace files will be contained.
Security Property SecurityProperty = username, instanceId, authcode, tftp ip address, tftp port, capf ip address, capf port, certificate path, security option, certificate status, fips compliant	NA	NA	NA	This field specifies the users security record (username, instanceId, authcode, tftp ip address, tftp port, capf ip address, capf port, certificate path, security option, certificate status, fips compliance), that will be stored in jtapi.ini files in a comma separated string. A semicolon separates the records. SecurityProperty = user, 123, 12345, 172.19.242.37, 3804, 172.19.242.37, 69, .\\, true, false, false; <next record>;…
Security Property Entries				
Username	NA	NA	NA	This field automatically populates the security profile of an application user who has previously configured a User Name/Instance ID pair and clicks any of the other edit boxes.
instanceId	NA	NA	NA	This field specifies the application instance identifier. If an application is connecting to CTIManager with the same user, it needs to define an Instance ID for each instance of the application to download the certificate Authorization String.

Jtapi.ini fields	Default	Min	Max	Description
authcode	NA	NA	NA	This field specifies authorization string that is configured in the Cisco Unified Communications Manager database. This can be used only once for getting certificate.
Communications Manager TFTP IP address	NA	NA	NA	This field specifies the TFTP Address of Cisco Unified Communications Manager (normally, the Cisco Unified Communications Manager IP Address)
CallManager TFTP port	69	NP	NP	This field displays the default value of the CallManager TFTP port.Do not change the default value of 69 unless advised to do so by the System Administrator.
Communications Manager CAPF IP server address	NA	NA	NA	This field specifies CAPF Server IP Address
Communications Manager CAPF server port	3804	NP	NP	This field displays the default value (3804) for CAPF server port. Be aware, you can configure this value in Cisco Unified Communications Manager Administration service parameters. Ensure that the value you enter through this interface should match the value configured on Cisco Unified Communications Manager Administration window.
Certificate path	JTAPI.jar location	NA	NA	This field specifies the location where application wants sever and client certificates to be installed. If this field is left blank, the system installs certificates in the ClassPath of JTAPI.jar
Enable secure connection	TRUE	NA	NA	This field, if set to TRUE then JTAPI will make a nonsecure connection to CTI even if certificates are updated/installed.
Certificate Update Status	NA	NA	NA	The JTAPI Preferences dialog box is used to configure the security profile for one or more User Name/Instance ID pairs.
FIPS Compliance	FALSE	NA	NA	This field, if set to TRUE, will enable the use of FIPS compliant cryptography algorithms and libraries in JTAPI.

Sample jtapi.ini File with Default Values

```
#Cisco Unified JTAPI version 7.0(1.1000)-1 Release ini parameters
#Wed Sep 14 16:55:30 PDT 2008
INFORMATIONAL = 0
DesiredServerHeartbeatInterval = 30
TracePath = .
FileNameExtension = log
SyslogCollector =
TraceFileSize = 1048576
UseAlarmService = 0
ProviderOpenRequestTimeout = 200
JtapiPostConditionTimeout = 15
ApplicationPriority = 2
SecurityTraceEnabled = 0
AlarmServicePort = 1444
RouteSelectTimeout = 5000
ProviderRetryInterval = 30
QueueStatsEnabled = 0
FileNameBase = CiscoJtapi
JTAPI_DEBUGGING = 0
PeriodicWakeupEnabled = 0
CTI_DEBUGGING = 0
JTAPINotificationPort = 2789
Traces = WARNING;INFORMATIONAL;DEBUG
PeriodicWakeupInterval = 50
AlarmServiceHostname =
QueueSizeThreshold = 25
Debugging = JTAPI_DEBUGGING;JTAPIIMPL_DEBUGGING;CTI_DEBUGGING;CTIIMPL_DEBUGGING;
PROTOCOL_DEBUGGING;MISC_DEBUGGING
PROTOCOL_DEBUGGING = 0
UseSystemDotOut = 0
MISC_DEBUGGING = 0
UseSameDirectory = 1
NumTraceFiles = 10
UseSyslog = 0
DEBUG = 0
SecurityTraceLevel = 0
UseTraceFile = 1
WARNING = 0
CMAssignedAppID = 0
UseProgressAsDisconnectedDuringErrorEnabled = 0
CtiManagers = ;;;;;;;;;
Directory =
CTIIMPL_DEBUGGING = 0
CtiRequestTimeout = 30
JTAPIIMPL_DEBUGGING = 0
SyslogCollectorUDPPort = 514
SecurityProperty = cisco, 123, 12345,  A.B.C.D, 3804,  A.B.C.D, 69,
/C\:/Program Files/JTAPITools/./,  false,  false;
```



CHAPTER 5

Cisco Unified JTAPI Extensions

The Cisco Unified JTAPI extension consists of a set of classes and interfaces that expose the additional functionality not readily exposed in JTAPI 1.2 specification but are available in Cisco Unified Communications Manager. Developers can use the extensions to create new applications or modify existing extensions to create new methods.

This chapter describes the extensions (interfaces and classes) that are available for implementation in a Cisco Unified Communications Manager.

Class Hierarchy

The following class hierarchy is contained in the com.cisco.jtapi.extensions package.

```
hierarchy.java.lang.Object
   com.cisco.jtapi.extensions.CiscoAddressCallInfo
   com.cisco.jtapi.extensions.CiscoJtapiVersion
   com.cisco.jtapi.extensions.CiscoMediaCapability
      com.cisco.jtapi.extensions.CiscoG711MediaCapability
      com.cisco.jtapi.extensions.CiscoG723MediaCapability
      com.cisco.jtapi.extensions.CiscoG729MediaCapability
      com.cisco.jtapi.extensions.CiscoGSMMediaCapability
      com.cisco.jtapi.extensions.CiscoWideBandMediaCapability
   com.cisco.jtapi.extensions.CiscoRTPParams
   java.lang.Throwable (implements java.io.Serializable)
      java.lang.Exception
    com.cisco.jtapi.extensions.CiscoRegistrationException
    com.cisco.jtapi.extensions.CiscoUnregistrationException
```

CiscoAddressCallInfo

Class History

Cisco Unified Communications Manager Release	Description
7.1 (2)	Added the history table to track changes.

Declaration

public class CiscoAddressCallInfo extends java.lang.Object

java.lang.Object

com.cisco.jtapi.extensions.CiscoAddressCallInfo

Constructors

CiscoAddressCallInfo (int inumActiveCalls, int imaxActiveCalls, int inumCallsOnHold, int imaxCallsOnHold)

CiscoAddressCallInfo (int inumActiveCalls, int imaxActiveCalls, int inumCallsOnHold, int imaxCallsOnHold, CiscoCall[] icalls)

Fields

None

Methods

Table 15: Methods in CiscoAddressCallInfo

Interface	Method	Description
CiscoCall[]	getCalls()	Returns the array of Cisco calls on the CiscoAddress.
CiscoCall[]	getTerminal()	Returns the terminal on which the address got activated (i.e. marked unrestricted)
int	getMaxActiveCalls()	Returns the maximum number of active calls supported on the CiscoAddress, as an integer.
int	getMaxCallsOnHold()	Returns the maximum number of calls that can be put on hold on the CiscoAddress, as an integer.
int	getNumActiveCalls()	Returns the number of active calls on the CiscoAddress, as an integer.
int	getNumCallsOnHold()	Returns the number of held calls on the CiscoAddress, as an integer.

Inherited Methods

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Related Documentation

None

CiscoG711MediaCapability

The CiscoG711MediaCapability object specifies the properties for a G.711 encoded RTP stream. Applications that support G.711 media termination use this object to specify their preferred packet size when registering a CiscoMediaTerminal. The default packet size is thirty milliseconds.

Class History

Cisco Unified Communications Manager Release	Description
7.1(x)	Added history table to track changes.

Declaration

public class CiscoG711MediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoG711MediaCapability

Constructors

Table 16: Constructors in CiscoG711MediaCapability

Interface	Constructor	Description
public	CiscoG711MediaCapability(intrtpPacketFrameSize)	Constructs a CiscoG711MediaCapability.
public	CiscoG711MediaCapability()	Constructs a CiscoG711MediaCapability.

Fields

Table 17: Fields in CiscoG711MediaCapability

Interface	Field	Description
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP Packet Framesize: Twenty millisecond RTP packet.
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP Packet Framesize: Thirty millisecond RTP packet.
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP Packet Framesize: Sixty millisecond RTP packet.

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

Methods

None

Inherited Methods

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See Constant Field Values, on page 1661.

CiscoG723MediaCapability

The CiscoG723MediaCapability object specifies the properties for a G.723 encoded RTP stream. Applications that support G.723 media termination use this object to specify their preferred packet size and bit rate when registering a CiscoMediaTerminal. The default packet size is thirty milliseconds and the default bit rate is 6.4k.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoG723MediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoG723MediaCapability

Constructors

Table 18: Constructors in CiscoG723MediaCapability

Interface	Constructor	Description
public	CiscoG723MediaCapability (intrtpPacketFrameSize, intbitRate)	Constructs a CiscoG723MediaCapability.

Fields

Table 19: Fields in CiscoG723MediaCapability

Interface	Field	Description
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP Packet Framesize: Twenty millisecond RTP packet.
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP Packet Framesize: Thirty millisecond RTP packet.
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP Packet Framesize: Sixty millisecond RTP packet.

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

Methods

Table 20: Methods in CiscoG723MediaCapability

Interface	Method	Description
public int	getBitRate()	Returns the bit rate specified by this capability object. Returns: a bit rate from the RTPBitRate interface.
public java.lang.String	toString()	Overwrites the Object.toString() method. Overrides: toString in class CiscoMediaCapability.

Inherited Methods

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See Constant Field Values, on page 1661.

CiscoG729MediaCapability

The CiscoG729MediaCapability object specifies the properties for a G.729 encoded RTP stream. Applications that support G.729 media termination use this object to specify their preferred packet size when registering a CiscoMediaTerminal. The default packet size is thirty milliseconds.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoG729MediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoG729MediaCapability

Constructors

Table 21: Constructors in G729MediaCapability

Constructor	Description
CiscoG729MediaCapability(int payload, int rtpPacketFrameSize)	Constructs a CiscoG729MediaCapability.

Fields

Table 22: Fields in CiscoG729MediaCapability

Interface	Fields	Description
staticint	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP Packet Framesize: Sixty millisecond RTP packet.
staticint	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP Packet Framesize: Thirty millisecond RTP packet.
staticint	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP Packet Framesize: Twenty millisecond RTP packet.

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

Methods

None

Inherited Methods

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See Constant Field Values, on page 1661.

CiscoGSMMediaCapability

The CiscoGSMMediaCapability object specifies the properties for a GSM encoded RTP stream. Applications that support GSM media termination use this object to specify their preferred packet size when registering a CiscoMediaTerminal. The default packet size is thirty milliseconds.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoGSMMediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoGSMMediaCapability

Constructors

Table 23: Constructors in CiscoGSMMediaCapability

Interface	Constructor	Description
public	CiscoGSMMediaCapability()	Constructs a CiscoGSMMediaCapability
public	CiscoGSMMediaCapability(int rtpPacketFrameSize)	Constructs a CiscoGSMMediaCapability.

Fields

Table 24: Fields in CiscoGSMMediaCapability

Interface	Field	Description
staticint	FRAMESIZE_EIGHTY_MILLISECOND_PACKET	RTP Packet Framesize: Eighty millisecond RTP packet

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

Methods

None

Inherited Methods

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

None

CiscoJtapiVersion

This class gives the version information of the installed Cisco JTAPI. Programs can get the version number using the accessor methods. Cisco Jtapi Version is in a.b(x.y) format where “a” indicates the major version, “b” indicates the minor version, “x” indicates the revision number, and “y” indicates the build number .

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoJtapiVersion extends java.lang.Object

java.lang.Object

com.cisco.jtapi.extensions.CiscoJtapiVersion

Constructors

publicCiscoJtapiVersion()None

Fields

None

Methods

Table 25: Methods in CiscoJtapiVersion

Interface	Method	Description
java.lang.String	getBuildDescription()	Returns “release” if it is a release version or debug if it is not a release version.
int	getBuildNumber()	Returns the build number of the version.
int	getExtendedBuildNumber()	Returns the extended build number of the version.
int	getMajorVersion()	Returns the major version number.
int	getMinorVersion()	Returns the minor version number.
int	getRevisionNumber()	Returns the revision number of the version.
public java.lang.String	getVersion()	Returns the version information in a.b(x.y)-z format without a name.
public java.lang.String	toString()	Returns the version information in a.b(x.y)-z format. Overrides toString in class java.lang.Object.

Inherited Methods

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

None

CiscoMediaCapability

The CiscoMediaCapability object specifies the properties of a particular media format that an application can support for CiscoMediaTerminals that it registers. Because CiscoMediaCapability is an abstract class, applications may only construct its subclasses directly.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoMediaCapability extends java.lang.Object

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

Subclasses

CiscoG711MediaCapability, CiscoG723MediaCapability, CiscoG729MediaCapability, CiscoGSMMediaCapability, CiscoWideBandMediaCapability

Constructors

Table 26: Constructors in CiscoMediaCapability

Interface	Constructor	Description
public	CiscoMediaCapability(intpayloadType, intmaxFramesPerPacket)	Constructs a CiscoMediaCapability object for the specified payload type and packet size (in milliseconds).

Fields

Table 27: Fields in CiscoMediaCapability

Interface	Field	Description
static	G711_64K_30_MILLISECONDS	G.711 capability with default parameters.
static	G723_6K_30_MILLISECONDS	G.723 capability with default parameters.
static	G729_30_MILLISECONDS	G.729 capability with default parameters.
static	GSM_80_MILLISECONDS	GSM capability with default parameters.
static	WIDEBAND_256K_10_MILLISECONDS	Wideband capability with default parameters.

Methods

Table 28: Methods in CiscoMediaCapability

Interface	Method	Description
int	getMaxFramesPerPacket(	Returns the packet size (in milliseconds) that this object specifies.The maxFramesPerPacket parameter is a carryover from the H.245 protocol definition. Cisco Unified Communications Manager does not use this field as the number of frames per RTP packet, but rather as the number of milliseconds of audio per RTP packet that the device can receive. Third-party IP phones may use different (higher) rates even though these rates may not be exceeded to and or from Cisco Unified IP phones.
int	getPayloadType()	Returns a payload type from the RTPPayload interface that this object specifies.
boolean	isSupported()	Returns whether the payload of this object is supported or not. True if the payloadType is supported, or otherwise false
java.lang.String	toString()	Overrides toString in class java.lang.Object.

Inherited Methods

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See CiscoG711MediaCapability, CiscoG723MediaCapability, CiscoG729MediaCapability, CiscoGSMMediaCapability, CiscoWideBandMediaCapability, CiscoRTPBitRate, and CiscoRTPPayload.

CiscoMultiMediaCapabilityInfo

CiscoMultiMediaCapabilityInfo interface contains the multimedia capabilities of a terminal. Applications can get the video capability, number of screens, and telepresence interoperability of the terminal using this API.

Declaration

public interface CiscoMultiMediaCapabilityInfo

com.cisco.jtapi.extensions.CiscoMultiMediaCapabilityInfo

Fields

Table 29: Fields in CiscoMultiMediaCapabilityInfo

Interface	Field	Description
static final int	NONE	Indicates that the CiscoMultiMediaCapabilityInfo.getVideoCapability () for this terminal is NONE.
static final int	VIDEO_ENABLED	CiscoMultiMediaCapabilityInfo.getVideoCapability () for this terminal is VIDEO_ENABLED.
static final int	TELEPRESENCEINTEROP_NONE	Indicates that the CiscoMultiMediaCapabilityInfo.getTelepresenceInfo() for this terminal is TELEPRESENCEINTEROP_NONE.
static final int	TELEPRESENCEINTEROP_ENABLED	CiscoMultiMediaCapabilityInfo. getTelepresenceInfo () for this terminal is TELEPRESENCEINTEROP_ENABLED

Methods

Table 30: Methods in MultiMediaCapabilityInfo

Interface	Method	Description
int	getVideoCapability()	Returns the video capability of the Terminal. The video capability can be NONE or VIDEO_ENABLED
int	getTelepresenceInfo()	Returns the telepresence capability of the Terminal. The telepresence capability can be TELEPRESENCEINTEROP_NONE or TELEPRESENCEINTEROP_ENABLED
int	getScreenCount()	Returns the number of screens present on the Terminal.

CiscoRegistrationException

The CiscoMediaTerminal.register method throws this exception when the registration process fails for any reason. For example, registration would fail if the Provider were OUT_OF_SERVICE or if the device were already registered.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoRegistrationException extends java.lang.Exception

java.lang.Object

java.lang.Throwable

java.lang.Exception

com.cisco.jtapi.extensions.CiscoRegistrationException

Implemented Interfaces

java.io.Serializable

Constructors

Table 31: Constructors in CiscoRegistrationException

Interface	Constructor	Description
public	CiscoRegistrationException (java.lang.Stringdescription)	Takes the description of the exception as a parameter.

Methods

None

Inherited Methods

From Class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See CiscoMediaTerminal.register(java.net.InetAddress, int, com.cisco.jtapi.extensions.CiscoMediaCapability[]).

CiscoRTPParams

You can use the CiscoRTPParams class to specify a dynamic RTP address and port number for a media terminal on a per-call basis. Applications can pass this object in setRTPParams() of CiscoMediaTerminal. These parameters are only valid for a particular call.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoRTPParams extends java.lang.Object

java.lang.Object

Constructors

CiscoRTPParams (java.net.InetAddress, rtpAddress, int rtpPort)

Fields

None

Methods

Table 32: Methods in CiscoRTPParams

Interface	Method	Description
java.net.InetAddress	getRTPAddress()	Returns the Internet address for the inbound RTP stream of the associated call.
java.lang.String	getRTPAddressHostName()	Returns the IP host name for the inbound RTP stream of the associated call.
byte[]	getRTPByteAddress()	Returns the Internet address in byte format for the inbound RTP stream.
int	getRTPPort()	Returns the UDP port for the inbound RTP stream.
java.lang.String	toString()	Returns a String in the format “IP address/port number.” Overrides toString in class java.lang.Object.

Inherited Methods

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See CiscoTerminal and CiscoMediaTerminal.

CiscoUnregistrationException

The CiscoMediaTerminal.unregister method throws this exception when the unregistration process fails. For example, registration fails if the Provider is OUT_OF_SERVICE or the Terminal is already unregistered.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoUnregistrationException extends java.lang.Exception

java.lang.Object

java.lang.Throwable

java.lang.Exception

com.cisco.jtapi.extensions.CiscoUnregistrationException

Implemented Interfaces

java.io.Serializable

Constructors

Table 33: Constructors in CiscoUnregistrationException

Interface	Constructor	Description
public	CiscoUnregistrationException ()(java.lang.Stringdescription)	None

Fields

None

Methods

None

Inherited Methods

From Class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See CiscoMediaTerminal.unregister(), Serialized Form.

CiscoWideBandMediaCapability

The CiscoWideBandMediaCapability object specifies the properties for a wide band encoded RTP stream. Applications that support wide band media termination use this object to specify their preferred packet size when registering a CiscoMediaTerminal. The default packet size is ten milliseconds.

Class History

Cisco Unified Communications Manager Release	Description
7.1x	Added history table to track changes.

Declaration

public class CiscoWideBandMediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoWideBandMediaCapability

Constructors

Table 34: Constructors in CiscoWideBandMediaCapability

Interface	Constructor	Description
public	CiscoWideBandMediaCapability(intpacketsize)	Constructs a CiscoWideBandMediaCapability object with the specified packet size. The default is ten–millisecond packet size. **Parameters** • packetsize—The RTP packet Framesize.

Fields

Table 35: Fields in CiscoWideBandMedicaCapability

Interface	Field	Description
staticint	FRAMESIZE_TEN_MILLISECOND_PACKET	RTP Packet Framesize: Ten millisecond RTP packet

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

Methods

None

Inherited Methods

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported, toString

From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Related Documentation

See Constant Field Values, on page 1661.

Interface Hierarchy

The following interface hierarchy is contained in the com.cisco.jtapi.extensions package hierarchy.

```
javax.telephony.Address
   com.cisco.jtapi.extensions.CiscoAddress (also extends
                                com.cisco.jtapi.extensions.CiscoObjectContainer)
      com.cisco.jtapi.extensions.CiscoIntercomAddress
   javax.telephony.callcenter.RouteAddress
      com.cisco.jtapi.extensions.CiscoRouteAddress


javax.telephony.AddressObserver
   com.cisco.jtapi.extensions.CiscoAddressObserver


javax.telephony.Call
   javax.telephony.callcontrol.CallControlCall
      com.cisco.jtapi.extensions.CiscoCall (also extends
                                com.cisco.jtapi.extensions.CiscoObjectContainer)
   com.cisco.jtapi.extensions.CiscoConsultCall


com.cisco.jtapi.extensions.CiscoCallCtlTermConnHeldReversionEv


com.cisco.jtapi.extensions.CiscoConferenceChain


com.cisco.jtapi.extensions.CiscoFeatureReason


com.cisco.jtapi.extensions.CiscoJtapiException


com.cisco.jtapi.extensions.CiscoJtapiProperties


com.cisco.jtapi.extensions.CiscoLocales


com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator


com.cisco.jtapi.extensions.CiscoMediaConnectionMode


com.cisco.jtapi.extensions.CiscoMediaEncryptionAlgorithmType


com.cisco.jtapi.extensions.CiscoMediaEncryptionKeyInfo


com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator


com.cisco.jtapi.extensions.CiscoMonitorInitiatorInfo


com.cisco.jtapi.extensions.CiscoMonitorTargetInfo


com.cisco.jtapi.extensions.CiscoObjectContainer
   com.cisco.jtapi.extensions.CiscoAddress (also extends javax.telephony.Address)
     com.cisco.jtapi.extensions.CiscoIntercomAddress
   com.cisco.jtapi.extensions.CiscoCall (also extends
                                        javax.telephony.callcontrol.CallControlCall)
```

```
   com.cisco.jtapi.extensions.CiscoConsultCall
   com.cisco.jtapi.extensions.CiscoCallID
   com.cisco.jtapi.extensions.CiscoConnection (also extends
                              javax.telephony.callcontrol.CallControlConnection)
   com.cisco.jtapi.extensions.CiscoConnectionID
   com.cisco.jtapi.extensions.CiscoConsultCall
   com.cisco.jtapi.extensions.CiscoIntercomAddress
   com.cisco.jtapi.extensions.CiscoJtapiPeer (also extends javax.telephony.JtapiPeer,
                                         com.cisco.services.tracing.TraceModule)
   com.cisco.jtapi.extensions.CiscoMediaTerminal
   com.cisco.jtapi.extensions.CiscoProvider
   com.cisco.jtapi.extensions.CiscoRouteTerminal
   com.cisco.jtapi.extensions.CiscoTerminal (also extends javax.telephony.Terminal)
      com.cisco.jtapi.extensions.CiscoMediaTerminal
      com.cisco.jtapi.extensions.CiscoRouteTerminal
   com.cisco.jtapi.extensions.CiscoTerminalConnection (also extends
                       javax.telephony.callcontrol.CallControlTerminalConnection)

com.cisco.jtapi.extensions.CiscoPartyInfo

com.cisco.jtapi.extensions.CiscoProvFeatureID

com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoRecorderInfo

com.cisco.jtapi.extensions.CiscoRTPBitRate

com.cisco.jtapi.extensions.CiscoRTPHandle

com.cisco.jtapi.extensions.CiscoRTPInputProperties

com.cisco.jtapi.extensions.CiscoRTPOutputProperties

com.cisco.jtapi.extensions.CiscoRTPPayload

com.cisco.jtapi.extensions.CiscoSynchronousObserver

com.cisco.jtapi.extensions.CiscoTermConnPrivacyChangedEv

com.cisco.jtapi.extensions.CiscoTermEvFilter

com.cisco.jtapi.extensions.CiscoTerminalProtocol

com.cisco.jtapi.extensions.CiscoTone

com.cisco.jtapi.extensions.CiscoUrlInfo

javax.telephony.Connection
   javax.telephony.callcontrol.CallControlConnection
      com.cisco.jtapi.extensions.CiscoConnection (also extends
                                  com.cisco.jtapi.extensions.CiscoObjectContainer)

javax.telephony.events.Ev
```

```
   javax.telephony.events.AddrEv
      com.cisco.jtapi.extensions.CiscoAddrEv (also extends
                                                  com.cisco.jtapi.extensions.CiscoEv)
       com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
       com.cisco.jtapi.extensions.CiscoAddrInServiceEv
       com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv
       com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv
       com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (also extends
                                com.cisco.jtapi.extensions.CiscoOutOfServiceEv)
       com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv
      javax.telephony.callcontrol.events.CallCtlEv
         javax.telephony.callcontrol.events.CallCtlCallEv (also extends
                                                            javax.telephony.events.CallEv)
       javax.telephony.callcontrol.events.CallCtlConnEv (also extends
                                                javax.telephony.events.ConnEv)
    javax.telephony.callcontrol.events.CallCtlConnOfferedEv
com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv
   javax.telephony.events.CallEv
      javax.telephony.events.CallActiveEv
       com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (also extends
                                  com.cisco.jtapi.extensions.CiscoCallEv)
         javax.telephony.callcontrol.events.CallCtlCallEv (also extends
                                javax.telephony.callcontrol.events.CallCtlEv)
       javax.telephony.callcontrol.events.CallCtlConnEv (also extends
                                                javax.telephony.events.ConnEv)
   javax.telephony.callcontrol.events.CallCtlConnOfferedEv
   com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv
       com.cisco.jtapi.extensions.CiscoCallEv (also extends
                                                  com.cisco.jtapi.extensions.CiscoEv)
       com.cisco.jtapi.extensions.CiscoCallChangedEv
       com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv
       com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv
       com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv
       com.cisco.jtapi.extensions.CiscoConferenceEndEv
       com.cisco.jtapi.extensions.CiscoConferenceStartEv
       com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (also extends
                                    javax.telephony.events.CallActiveEv)
       com.cisco.jtapi.extensions.CiscoToneChangedEv
       com.cisco.jtapi.extensions.CiscoTransferEndEv
       com.cisco.jtapi.extensions.CiscoTransferStartEv
   javax.telephony.events.ConnEv
      javax.telephony.callcontrol.events.CallCtlConnEv (also extends
                               javax.telephony.callcontrol.events.CallCtlCallEv)
   javax.telephony.callcontrol.events.CallCtlConnOfferedEv
   com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv
   javax.telephony.events.TermConnEv
      com.cisco.jtapi.extensions.CiscoTermConnMonitoringEndEv
      com.cisco.jtapi.extensions.CiscoTermConnMonitoringStartEv
      com.cisco.jtapi.extensions.CiscoTermConnMonitorInitiatorInfoEv
      com.cisco.jtapi.extensions.CiscoTermConnMonitorTargetInfoEv
      com.cisco.jtapi.extensions.CiscoTermConnRecordingEndEv
      com.cisco.jtapi.extensions.CiscoTermConnRecordingStartEv
      com.cisco.jtapi.extensions.CiscoTermConnRecordingTargetInfoEv
      com.cisco.jtapi.extensions.CiscoTermConnSelectChangedEv


com.cisco.jtapi.extensions.CiscoEv
   com.cisco.jtapi.extensions.CiscoAddrActivatedEv
   com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
   com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv
   com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
   com.cisco.jtapi.extensions.CiscoAddrCreatedEv
   com.cisco.jtapi.extensions.CiscoAddrEv (also extends
                                                  javax.telephony.events.AddrEv)
      com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
```

```
      com.cisco.jtapi.extensions.CiscoAddrInServiceEv
      com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv
      com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv
      com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (also extends
                                    com.cisco.jtapi.extensions.CiscoAddrEv,
                                com.cisco.jtapi.extensions.CiscoOutOfServiceEv)
      com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv
      com.cisco.jtapi.extensions.CiscoAddrInServiceEv
      com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv
      com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv
      com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (also extends
                                     com.cisco.jtapi.extensions.CiscoAddrEv)
      com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv
   com.cisco.jtapi.extensions.CiscoAddrRemovedEv
   com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
   com.cisco.jtapi.extensions.CiscoCallChangedEv
   com.cisco.jtapi.extensions.CiscoCallEv (also extends
                                                        javax.telephony.events.CallEv)
      com.cisco.jtapi.extensions.CiscoCallChangedEv
      com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv
      com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv
      com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv
      com.cisco.jtapi.extensions.CiscoConferenceEndEv
      com.cisco.jtapi.extensions.CiscoConferenceStartEv
      com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (also extends
                                          javax.telephony.events.CiscoCallEv)
      com.cisco.jtapi.extensions.CiscoToneChangedEv
      com.cisco.jtapi.extensions.CiscoTransferEndEv
      com.cisco.jtapi.extensions.CiscoTransferStartEv


com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv


com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv


com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv


com.cisco.jtapi.extensions.CiscoConferenceEndEv


com.cisco.jtapi.extensions.CiscoConferenceStartEv


com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (also extends
                                        javax.telephony.events.CallActiveEv,
                                           com.cisco.jtapi.extensions.CiscoCallEv)


com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv


com.cisco.jtapi.extensions.CiscoOutOfServiceEv
   com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (also extends
                                     com.cisco.jtapi.extensions.CiscoAddrEv)
   com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv (also extends
                                     com.cisco.jtapi.extensions.CiscoTermEv)


com.cisco.jtapi.extensions.CiscoProvCallParkEv


com.cisco.jtapi.extensions.CiscoProvFeatureEv (also extends
                                                        javax.telephony.events.ProvEv)
```

```
com.cisco.jtapi.extensions.CiscoAddrActivatedEv
com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv
com.cisco.jtapi.extensions.CiscoAddrCreatedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoProvFeatureEv
   com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
   com.cisco.jtapi.extensions.CiscoTermActivatedEv
   com.cisco.jtapi.extensions.CiscoTermCreatedEv
   com.cisco.jtapi.extensions.CiscoTermRemovedEv
   com.cisco.jtapi.extensions.CiscoTermRestrictedEv
   com.cisco.jtapi.extensions.CiscoProvFeatureEv
   com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
   com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
com.cisco.jtapi.extensions.CiscoTermActivatedEv
com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
com.cisco.jtapi.extensions.CiscoTermCreatedEv
com.cisco.jtapi.extensions.CiscoTermDataEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv
com.cisco.jtapi.extensions.CiscoTermEvFilter (also extends
                                                 javax.telephony.events.TermEv)
   com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv
   com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
   com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
   com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
   com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
   com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
   com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
   com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
   com.cisco.jtapi.extensions.CiscoTermDataEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
   com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv
   com.cisco.jtapi.extensions.CiscoTermInServiceEv
   com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv(also extends
                           com.cisco.jtapi.extensions.CiscoOutOfServiceEv)
   com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotEv
com.cisco.jtapi.extensions.CiscoTermInServiceEv
com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv (also extends
```

```
                                com.cisco.jtapi.extensions.CiscoOutOfServiceEv,
                                                    com.cisco.jtapi.extensions.CiscoTermEv)
   com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv
   com.cisco.jtapi.extensions.CiscoTermRemovedEv
   com.cisco.jtapi.extensions.CiscoTermRestrictedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotEv
   com.cisco.jtapi.extensions.CiscoToneChangedEv
   com.cisco.jtapi.extensions.CiscoTransferEndEv
   com.cisco.jtapi.extensions.CiscoTransferStartEv


javax.telephony.events.ProvEv
   com.cisco.jtapi.extensions.CiscoProvEv (also extends
                                                    com.cisco.jtapi.extensions.CiscoEv)
      com.cisco.jtapi.extensions.CiscoAddrActivatedEv
      com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
      com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
      com.cisco.jtapi.extensions.CiscoAddrCreatedEv
      com.cisco.jtapi.extensions.CiscoAddrRemovedEv
      com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv
      com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
      com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
      com.cisco.jtapi.extensions.CiscoProvCallParkEv
      com.cisco.jtapi.extensions.CiscoProvFeatureEv
    com.cisco.jtapi.extensions.CiscoProvCallParkEv
      com.cisco.jtapi.extensions.CiscoRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
      com.cisco.jtapi.extensions.CiscoTermActivatedEv
      com.cisco.jtapi.extensions.CiscoTermCreatedEv
      com.cisco.jtapi.extensions.CiscoTermRemovedEv
      com.cisco.jtapi.extensions.CiscoTermRestrictedEv


javax.telephony.events.TermEv
   com.cisco.jtapi.extensions.CiscoTermEv (also extends
                                                    com.cisco.jtapi.extensions.CiscoEv)
   com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv
   com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
   com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
   com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
   com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
   com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
   com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
   com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
   com.cisco.jtapi.extensions.CiscoTermDataEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv
   com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
   com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv
   com.cisco.jtapi.extensions.CiscoTermInServiceEv
   com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv (also extends
                                       com.cisco.jtapi.extensions.CiscoOutOfServiceEv)
   com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv
   com.cisco.jtapi.extensions.CiscoTermSnapshotEv


javax.telephony.JtapiPeer
   com.cisco.jtapi.extensions.CiscoJtapiPeer (also extends
                               com.cisco.jtapi.extensions.CiscoObjectContainer,
                                                    com.cisco.services.tracing.TraceModule)
```

```
javax.telephony.Provider
   com.cisco.jtapi.extensions.CiscoProvider (also extends
                                com.cisco.jtapi.extensions.CiscoObjectContainer)


javax.telephony.capabilities.ProviderCapabilities
   com.cisco.jtapi.extensions.CiscoProviderCapabilities


javax.telephony.ProviderObserver
   com.cisco.jtapi.extensions.CiscoProviderObserver


javax.telephony.callcenter.RouteSession
   com.cisco.jtapi.extensions.CiscoRouteSession


javax.telephony.callcenter.events.RouteSessionEvent
   javax.telephony.callcenter.events.RouteEvent
      com.cisco.jtapi.extensions.CiscoRouteEvent
   javax.telephony.callcenter.events.RouteUsedEvent
      com.cisco.jtapi.extensions.CiscoRouteUsedEvent


javax.telephony.Terminal
   com.cisco.jtapi.extensions.CiscoTerminal (also extends
                           com.cisco.jtapi.extensions.CiscoObjectContainer)
      com.cisco.jtapi.extensions.CiscoMediaTerminal
      com.cisco.jtapi.extensions.CiscoRouteTerminal


javax.telephony.TerminalConnection
   javax.telephony.callcontrol.CallControlTerminalConnection
      com.cisco.jtapi.extensions.CiscoTerminalConnection (also extends
com.cisco.jtapi.extensions.CiscoObjectContainer)


javax.telephony.TerminalObserver
   com.cisco.jtapi.extensions.CiscoTerminalObserver


com.cisco.services.tracing.TraceModule
   com.cisco.jtapi.extensions.CiscoJtapiPeer (also extends
                              com.cisco.jtapi.extensions.CiscoObjectContainer,
                                                    javax.telephony.JtapiPeer)
```

CiscoAddrActivatedEv

If an address is controlled and the restriction status changes to active, the system sends the CiscoAddrActivatedEv event to the application. Applications see this event whenever an Address or associated Terminal is in the control list. If any observers exist on the address already, applications see CiscoAddrInServiceEv. If no observers are present, applications can try to add observers, and the address will go in service.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrActivatedEv extends CiscoProvEv

Fields

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 36: Methods in CiscoAddrActivatedEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the Address which is activated.

Inherited Methods

From Interface javax.telephony.events.ProvEv

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

Superinterfaces

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlConnEv, javax.telephony.callcontrol.events.CallCtlConnOfferedEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.events.CallEv, javax.telephony.events.ConnEv, javax.telephony.events.Ev

Declaration

public interface CiscoCallCtlConnOfferedEv extends javax.telephony.callcontrol.events.CallCtlConnOfferedEv

Fields

None

Inherited Fields

From Interface javax.telephony.callcontrol.events.CallCtlConnOfferedEv

None

From Interface javax.telephony.callcontrol.events.CallCtlEv

CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 37: Methods in CiscoCallCtlConnOfferedEv

Interface	Method	Description
java.net.InetAddress	getCallingPartyIpAddr()	Returns the IP address of the calling party, or 0 (or null) if the IP Address is not available.

Inherited Methods

From Interface javax.telephony.callcontrol.events.CallCtlCallEv

getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

From Interface javax.telephony.callcontrol.events.CallCtlEv

getCallControlCause

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ConnEv

getConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

None

CiscoAddrActivatedOnTerminalEv

The CiscoAddrActivatedOnTerminalEv event gets sent when a shared line gets activated or a Terminal which has shared line gets activated.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrActivatedOnTerminalEv extends CiscoProvEv

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 38: Methods in CiscoAddrActivatedOnTerminalEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the address that is marked unrestricted on the terminal.
javax.telephony.Terminal	getTerminal()	Returns the terminal on which the address got activated (i.e. marked unrestricted).

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrAddedToTerminalEv

The system sends CiscoAddrAddedToTerminalEv when:

- A user adds a Terminal into the control list that contains a shared line, the system sends this event to the application. If a user has an address in the control list, and you add a new Terminal with the same address in control list, this event gets sent.
- An Extension Mobility (EM) user logs into a Terminal with a profile that contains a shared line, this event notifies that a new Terminal has been added to an already existing address.
- A new shared line gets added to a Terminal in a user control list, the system sends this event to the application.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrAddedToTerminalEv extends CiscoProvEv

Fields

Table 39: Fields in CiscoAddrAddedToTerminalEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL,

CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 40: Methods in CiscoAddrAddedToTerminalEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the address on which the new terminal is added.
javax.telephony.Terminal	getTerminal()	Returns the terminal that gets added to the Address.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrAutoAcceptStatusChangedEv

The system sends CiscoAddrAutoAcceptStatusChangedEv to applications whenever the AutoAccept status for the Address on the Terminal changes. If an Address has multiple Terminals, this event gets sent for the Address AutoAccept status on each individual Terminal.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrAutoAcceptStatusChangedEv extends CiscoAddrEv

Fields

Table 41: Fields in CiscoAddrAutoAcceptStatusChangedEv

Interface	Field
static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUTUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_R_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 42: Methods for CiscoAddrAutoAcceptStatusChangedEv

Interface	Method	Description
int	getAutoAcceptStatus()	Returns the AutoAccept Status of the Address on the Terminal. Returns CiscoAddress.AUTOACCEPT_OFF or CiscoAddress.AUTOACCEPT_ON
CiscoTerminal	getTerminal()	Returns the Terminal at which the AutoAccept status for this address is changing.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See getAutoAcceptStatus and CiscoAddress.getAutoAcceptStatus(Terminal terminal).

CiscoAddrCreatedEv

The CiscoAddrCreatedEv event gets sent when an Address gets added to the provider domain.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrCreatedEv extends CiscoProvEv

Fields

Table 43: Fields in CiscoAddrCreatedEv

Interface	Field
ID	static final int ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 44: Methods in CiscoAddrCreatedEv

Interface	Method	Description
getAddress	javax.telephony.Address getAddress()	Returns the address which got added to the provider domain. Returns the address that is added to the provider domain.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoAddrMonitorTerminatedEv

When a monitor session is terminated, the Supervisor who had initiated the session will be notified with this event.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New interface

Declaration

pubic interface CiscoAddrMonitorTerminatedEv extends CiscoAddrEv

Methods

Table 45: Methods in CiscoAddrMonitorTerminatedEv

Interface	Method	Description
Int	getTransactionID()	Returns the transaction ID for the session termination.
Address	getMonitorTargetAddress()	Returns the target address that was being monitored.
String	getMonitorTargetDevieName()	Returns the monitored device name.
Int	getMonitorTargetCalllegHandle()	Returns the call leg identifier for the monitored target.

Interface	Method	Description
String	getMonitorInitiatorDeviceName()	Returns the device name for the device that initiated the monitoring session.
Int	getCause()	Returns the reason that the monitoring session was terminated.

Related Documentation

CiscoAddress

The CiscoAddress interface extends the Address interface with additional Cisco Unified Communications Manager capabilities.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1, 2)	Added voice and fax message counts for the Enhanced Message Waiting Indication (MWI) feature for supported phones only.
7.1(3)	Updated for Terminal and Address Capability settings changes.
8.0(1)	Enhanced with the following: • New APIs getPickupGroup() to enable applications to get information about the Pickup Group the Address belongs to • New address type to indicate that the address represents hunt pilot. • New field that will represent a new kind of recording type, device-based recording.
9.0(1)	A new constant, SELECTIVE_RECORDING, is added. Two constants, APPLICATION_CONTROLLED_RECORDING, and DEVICE_CONTROLLED_RECORDING, are deprecated. Applications that upgrade to Release 9.0 or later releases should use the new SELECTIVE_RECORDING constant and not the deprecated APPLICATION_CONTROLLED_RECORDING and DEVICE_CONTROLLED_RECORDING constants. In Release 9.0 or later releases Unified CM and JTAPI never return the DEVICE_CONTROLLED_RECORDING constant.
10.0(1)	Enhanced with the following: • New APIs to create a persistent call and to retrieve the connection object associated to the persistent call. • a new API to create an announcement call in order to play announcements to the remote destinations.

Superinterfaces

javax.telephony.Address, CiscoObjectContainer, on page 468

Subinterfaces

CiscoIntercomAddress

Fields

Table 46: Fields in CiscoAddress

Interface	Field	Description
Static int	APPLICATION_CONTROLLED_RECORDING	Application controlled Recording is configured on the Address.
Static int	AUTO_RECORDING	Auto Recording is configured on the Address.
Static int	AUTOANSWER_OFF	AutoAnswer is off.
Static int	AUTOANSWER_UNKNOWN	AutoAnswer status is unknown.
Static int	AUTOANSWER_WITHHEADSET	AutoAnswer is allowed with a headset.
static int	AUTOANSWER_WITHSPEAKERSET	AutoAnswer is allowed with a speaker set.
public static final int	DEVICE_CONTROLLED_RECORDING	This value will be used to specify a new recording type. This type is used when the recording profile is configured on the device, and is thus “device controlled”
static int	EXTERNAL	This represents an external address with a valid name.
static int	EXTERNAL_UNKNOWN	This represents an external address with an unknown name.
static int	IN_SERVICE	The address is in service.
static int	INTERNAL	This is an internal address.
static int	MONITORING_TARGET	This represents an address with a monitoring target or agent.
static int	NO_RECORDING	Recording is off on the Address.
static int	OUT_OF_SERVICE	The address is out-of-service.
static int	RINGER_DEFAULT	Sets the ringer status to the configured value.
static int	RINGER_DISABLE	Disables the ringer for the address.

Interface	Field	Description
static int	RINGER_ENABLE	Enables the ringer for the address.
static int	SELECTIVE_RECORDING	This constant is added to replace the deprecated constants APPLICTION_CONTROLLED_RECORDING and DEVICE_CONTROLLED_RECORDING
static int	UNKNOWN	This represents an address with an unknown name.

Methods

Table 47: Methods in CiscoAddress

Interface	Method	Description
void	clearCallConnections ()	Use this interface to clear any phantom calls on the address. **Throws** javax. telephony. PrivilegeViolationException—Use this interface to clear any phantom calls on the address.
CiscoCall	createPersistentCall (Terminal terminal, String callerIDNumber, String callerIDName)	This interface creates a persistent call for this address and will return the call object for the newly created call. Note that CiscoProvider and the address must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. This API cannot be invoked on external addresses. Doing so will result in MethodNotSupportedException to be thrown. If while trying to allocate a globalCallId for the persistent call and an error occurs, ResourceUnavailableException will be thrown. All other errors encountered will result in PlatformException to be thrown.
	startAnnouncement (Terminal terminal, String announcementID)	This interface creates an announcement call for this address in order to play announcements to the remote destination. It returns the call object for the newly created call. Note that CiscoProvider and the address must be in IN_SERVICE state, otherwise InvalidStateException is thrown. This API cannot be invoked on external addresses. Doing so results in MethodNotSupportedException being thrown. If while trying to allocate a globalCallId for the announcement call and an error occurs, ResourceUnavailableException ise thrown. All other errors encountered results in PlatformException being thrown.
CiscoAddressCallInfo	getAddressCallInfo (javax. telephony. Terminal terminal)	Use this interface to get information about calls that are present at the Terminal.

Interface	Method	Description
String	getAsciiLabel (Terminal term)	This method returns the ASCII label configured for this address on Terminal term. Throws InvalidStateException, MethodNotSupportedException, InValidParameterException.
int	getAutoAcceptStatus (javax. telephony. Terminal terminal)	Returns the AutoAccept status of the Address on the Terminal. **Throws** javax. telephony. PlatformException, javax. telephony. InvalidStateException, javax. telephony. MethodNotSupportedException Returns the AutoAccept status of the Address on the Terminal. It may return one of the following constants: • CiscoAddress. AUTOACCEPT_OFF • CiscoAddress. AUTOACCEPT_ON **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE (getState () = = IN_SERVICE **Parameters** • terminal - The Terminal on which the AutoAccepts

Interface	Method	Description
int	getAutoAnswerStatus (javax. telephony. Terminal term)	This interface returns the AutoAnswer status of this Address on given Terminal. **Throws** javax. telephony. PlatformException, javax. telephony. InvalidStateException, javax. telephony. MethodNotSupportedException If return value is AUTOANSWER_OFF, that means AutoAnswer is disabled. If return value is AUTOANSWER_WITHHEADSET, that means AutoAnswer is enabled with HEADSET. If return value is AUTOANSWER_WITHSPEAKERSET, that means AutoAnswer is enabled with SPEAKERSET. If return value is AUTOANSWER_UNKNOWN, that means AutoAnswer status is UNKNOWN. **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE (getState () = = IN_SERVICE **Parameters** • term - Terminal at which AutoAnswer is checked Returns one of the following values: • CiscoAddress. AUTOANSWER_OFF • CiscoAddress. AUTOANSWER_WITHHEADSET • CiscoAddress. AUTOANSWER_WITHSPEAKERSET • CiscoAddress. AUTOANSWER_UNKNOWN **Throws** javax. telephony. InvalidStateException - The Provider or Address is not"IN_SERVICE". javax. telephony. PlatformException - If Address is not on Terminal term javax. telephony. MethodNotSupportedException - If Address is an External Address
int	getBusyTrigger (Terminal term)	This method returns the busy trigger configured for this address on terminal term. Throws InvalidStateException, InvalidArgumentException, MethodNotSupportedException.

Interface	Method	Description
int	getButtonPosition (Terminal term)	This method returns the button position of the address on terminal term. Throws InvalidStateException, InvalidArgumentException, MethodNotSupportedException.
javax. telephony. Terminal[]	getInServiceAddrTerminals ()	Use this interface to find out which Shared Lines are in service. In Shared Lines, the same Address appears on different Terminals. Returns: Terminal[]—An array of Terminals on which the Address is in service.
int	getMaxCalls (Terminal term)	This new method returns the maximum calls configured for an address on a terminal. This method throws InvalidStateException if the associated terminal is not registered to Cisco Unified Communication Manager. It throws InvalidArgumentException if terminal does not have this address. MethodNotSupportedException is be thrown if address is not in Provider
java. lang. String	getPartition ()	It returns the partition associated with an Address.
Connection	getPersistentConnection (Terminal terminal)	This interface will return the connection object that is associated with the persistent call. It returns null if there is no persistent call. This API cannot be invoked on external addresses. Doing so will result in MethodNotSupportedException to be thrown.
CiscoPickupGroup	getPickupGroup ()	This method returns a CiscoPickupGroup object that represents the Pickup Group DN and Partition that this Address belongs to.

Interface	Method	Description
int	getRecordingConfig (javax. telephony. Terminal term)	Returns the configured recording type on this Address. **Throws** javax. telephony. PlatformException, javax. telephony. InvalidStateException, javax. telephony. MethodNotSupportedException **Returns** • int—The configured recording type on this Address. • CiscoAddess. NO_RECORDING —The call cannot be recorded. • CiscoAddress. AUTO_RECORDING—Cisco Unified Communications Manager records all answered calls to/from this address. • CiscoAddress. APPLICATION_CONTROLLED_RECORDING—Calls get recorded only when the application initiates recording. Throws javax. telephony. InvalidStateException - The Provider or Address is not"IN_SERVICE". javax. telephony. PlatformException - If Address is not on Terminal term javax. telephony. MethodNotSupportedException - If Address is an External Address
int	getRegistrationState ()	**Deprecated**. This method has been replaced by the getState () method. Returns the state of this address can be any of the following constants: • CiscoAddress. OUT_OF_SERVICE • CiscoAddress. IN_SERVICE
javax. telephony. Terminal[]	getRestrictedAddrTerminals ()	Returns the array of Terminals on which this Address is restricted. In shared lines, few lines on Terminals may be restricted. Applications cannot see any call events for restricted Addresses. If a restricted Address is involved in a call with any other controlled Terminal, the system creates a Connection for the restricted Address, but there is not any TerminalConnection for the restricted Address. Returns: Terminal[]—An array of Terminals on which this Address is restricted. If none is restricted, this method returns null.

Interface	Method	Description
int	getState ()	Returns the state of this address. The state may be any of the following constants: • CiscoAddress. OUT_OF_SERVICE • CiscoAddress. IN_SERVICE
int	getType ()	Returns the following address constants: • CiscoAddress. INTERNAL • CiscoAddress. EXTERNAL • CiscoAddress. EXTERNAL_UNKNOWN • CiscoAddress. UNKNOWN • CiscoAddress. MONITORING_TARGET • CiscoAddress. HUNT_PILOT, if address is in a CiscoHuntConnection. • CiscoAddress. HUNT_PILOT, if address represents hunt pilot.
String	getUnicodeLabel (Terminal term)	This method returns the Unicode label configured for this address on Terminal term. Throws InvalidStateException, MethodNotSupportedException, InValidParameterException.
int	getVoiceMailPilot ()	This method returns the voice mail pilot of the address. Throws InvalidStateException, MethodNotSupportedException.
boolean	isRestricted (javax. telephony. Terminal terminal)	This method returns true if this Address on Terminal is restricted. ; false if not restricted.

Interface	Method	Description
void	setAutoAcceptStatus (int autoAcceptStatus, javax. telephony. Terminal terminal)	This method lets an application enable AutoAccept for this Address on CiscoMediaTerminal and/or CiscoRouteTerminal. Addresses on CiscoTerminal other than CiscoMediaTerminal or CiscoRouteTerminal will always have AutoAccept on. If the Terminal passed in the parameter is not a CiscoMediaTerminal or CiscoRouteTerminal, this method throws an exception. For a CiscoMediaTerminal that shares an Address with CiscoTerminal, Cisco recommends enabling AutoAccept on CiscoMediaTerminal. **Throws** javax. telephony. PlatformException, javax. telephony. InvalidStateException, javax. telephony. MethodNotSupportedException **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE (getState () = = IN_SERVICE **Post-conditions** Enables or Disables auto accept status **Parameters** • autoAcceptStatus - can be either CiscoAddress. AUTOACCEPT_OFF or CiscoAddress. AUTOACCEPT_ON. If autoAcceptStatus is AUTOACCEPT_ON, it will enable AutoAccept for Address on Terminal. If autoAcceptStatus is AUTOACCEPT_OFF, it will disable AutoAccept for Address on Terminal. • terminal - The Terminal on which AutoAccept will be enabled **Throws** javax. telephony. InvalidStateException - The Provider or Address is not "In_Service". javax. telephony. PlatformException - The Terminal does not have this Address. javax. telephony. MethodNotSupportedException - If the Terminal is not CiscoMediaTerminal or CiscoRouteTerminal.

Interface	Method	Description
void	setMessageWaiting (java. lang. String destination, boolean enable)	Specifies whether the message-waiting indicator should be activated or deactivated for the Address specified by the destination. If enable is true, message-waiting gets activated if not already activated. If enable is false, message-waiting gets deactivated if not already deactivated. **Throws** javax. telephony. MethodNotSupportedException, javax. telephony. InvalidStateException, javax. telephony. PrivilegeViolationException **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE **Post-conditions** Enables or disables the Message Waiting Indicator depending on the enable status. **Note** This implementation currently does not enforce the post-conditions as specified in CallControlAddress as follows: this. getMessageWaiting () = = enable CallCtlAddrMessageWaitingEv gets delivered for this Address. **Parameters** • destination - DN/Address message-waiting indicator is activated/deactivated • enable - True to activate message-waiting, false to deactivate **Throws** • javax. telephony. MethodNotSupportedException—This method is not supported by the given implementation. javax. telephony. InvalidStateException **Note** The Provider is not“in service. ” javax. telephony. PrivilegeViolationException **Note** The Provider user has insufficient privileges to invoke the message-waiting indicator for this destination.

Interface	Method	Description
void	setRingerStatus (int status)	Changes the ringer status on this address. **Throws** javax. telephony. MethodNotSupportedException, javax. telephony. InvalidStateException, javax. telephony. InvalidArgumentException Accepts one of the following constants: • CiscoAddress. RINGER_DEFAULT • CiscoAddress. RINGER_DISABLE • CiscoAddress. RINGER_ENABLE

Interface	Method	Description
void	setMessageSummary (boolean enable, boolean voiceCounts, int totalNewVoiceMsgs, int totalOldVoiceMsgs, boolean highPriorityVoiceCounts, int newHighPriorityVoiceMsgs, int oldHighPriorityVoiceMsgs, boolean faxCounts, int totalNewFaxMsgs, int totalOldFaxMsgs, boolean highPriorityFaxCounts, int newHighPriorityFaxMsgs, int oldHighPriorityFaxMsgs)	

Interface	Method	Description
		Use this interface to set the message-waiting indicator along with voice/fax message waiting counts If enable is true, message-waiting gets activated if not already activated. If enable is false, message-waiting gets deactivated if not already deactivated. **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE **Post-conditions** Enables or disables the Message Waiting Indicator and sets message waiting counts. **Parameters** • enable - True to activate message-waiting, false to deactivate • voiceCounts - indicates if voice message counts are provided • totalNewVoiceMsgs - specifies the total number of new voice messages waiting • totalOldVoiceMsgs - specifies the total number of old voice messages waiting • highPriorityVoiceCounts - indicates if high priority voice message counts are provided • newHighPriorityVoiceMsgs - specifies the number of new high priority voice messages waiting • oldHighPriorityVoiceMsgs - specifies the number of old high priority voice messages waiting • faxCounts - indicates if fax message counts are provided • totalNewFaxMsgs - specifies the total number of new fax messages waiting • totalOldFaxMsgs - specifies the total number of old fax messages waiting • highPriorityFaxCounts - indicates if high priority fax message counts are provided • newHighPriorityFaxMsgs - specifies the number of new high priority fax messages waiting • oldHighPriorityFaxMsgs - specifies the number of old high priority fax messages waiting **Throws** javax. telephony. MethodNotSupportedException - This method is not supported by the given implementation. javax. telephony. InvalidStateException - The Provider is not "in service. "

Interface	Method	Description
		javax. telephony. PrivilegeViolationException - The Provider user has insufficient privileges to set the message-waiting indicator or message counts for this destination.
void	setMessageSummary (java. lang. String destination, boolean enable, boolean voiceCounts, int totalNewVoiceMsgs, int totalOldVoiceMsgs, boolean highPriorityVoiceCounts, int newHighPriorityVoiceMsgs, int oldHighPriorityVoiceMsgs, boolean faxCounts, int totalNewFaxMsgs, int totalOldFaxMsgs, boolean highPriorityFaxCounts, int newHighPriorityFaxMsgs, int oldHighPriorityFaxMsgs)	Use this interface to set the message-waiting indicator along with voice/fax message waiting counts for the Address specified by the destination **Pre-conditions** (this. getProvider ()). getState () = = Provider. IN_SERVICE **Post-conditions** Enables or disables the Message Waiting Indicator and sets message waiting counts. **Parameters** • destination - DN/Address whose message-waiting indicator should be activated/deactivated • enable - True to activate message-waiting, false to deactivate • voiceCounts - indicates if voice message counts are provided • totalNewVoiceMsgs - specifies the total number of new voice messages waiting • totalOldVoiceMsgs - specifies the total number of old voice messages waiting • highPriorityVoiceCounts - indicates if high priority voice message counts are provided • newHighPriorityVoiceMsgs - specifies the number of new high priority voice messages waiting • oldHighPriorityVoiceMsgs - specifies the number of old high priority voice messages waiting • faxCounts - indicates if fax message counts are provided • totalNewFaxMsgs - specifies the total number of new fax messages waiting • totalOldFaxMsgs - specifies the total number of old fax messages waiting • highPriorityFaxCounts - indicates if high priority fax message counts are provided • newHighPriorityFaxMsgs - specifies the number of new high priority fax messages waiting • oldHighPriorityFaxMsgs - specifies the number of old high priority fax messages waiting

Inherited Methods

From Interface javax.telephony.Address

addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Parameters

- Terminal terminal: The terminal object you want to create the persistent call for.
- String callerIDNumber: The number you wish to show up on the remote destination's Caller ID.
- String callerIDName: The name you wish to show up on the remote destination's Caller ID.

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddressObserver

Applications implement this interface to receive CiscoAddrEv events such as CiscoAddrInServiceEv 0 CiscoAddrOutOfServiceEv when observing Addresses via the Address.addObserver method.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.AddressObserver

Declaration

public interface CiscoAddressObserver extends javax.telephony.AddressObserver

Fields

None

Methods

None

Inherited Methods

From Interface javax.telephony.AddressObserver

addressChangedEvent

Related Documentation

See CiscoAddrInServiceEv, CiscoAddrOutOfServiceEv for more information.

CiscoAddrEv

The CiscoAddrEv interface extends the JTAPI core javax.telephony.events.AddrEv interface and serves as the base interface for all Cisco extended JTAPI Address events. Every Address related event in this package extends this interface, directly or indirectly.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoEv, javax.telephony.events.Ev

Subinterfaces

CiscoAddrAutoAcceptStatusChangedEv, CiscoAddrInServiceEv, CiscoAddrIntercomInfoChangedEv, CiscoAddrIntercomInfoRestorationFailedEv, CiscoAddrOutOfServiceEv, CiscoAddrRecordingConfigChangedEv

Declaration

public interface CiscoAddrEv extends CiscoEv, javax.telephony.events.AddrEv

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See javax.telephony.events.AddrEv for more information.

CiscoAddrEvFilter

CiscoAddrEvFilter provided for applications to set filters for address events. The application can use the following APIs to enable/disable the filters to receive the event notifications on address or to check the value

set of the filter. Application can enable the filter, if it wishes to receive the new event (CiscoAddrParkStatusEv), for the rest of the events the filter values are true by default.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Added this event for Park Monitoring and Assisted DPark Support feature.
7.1.(3)	Interface is enhanced to allow application set filter on address to enable and disable CiscoAddrVoiceMailPilotChangedEv.
8.0(1)	Enhanced with the following: • getCiscoAddrMonitoringTerminatedEvFilter() • setCiscoAddrMonitoringTerminatedEvFilter() By default the filter will be set to 'true' and CiscoMonitoringTerminatedEv will be delivered. To stop receiving this event applications need to set the filter to false.

Fields

None

Methods

Table 48: Methods in CiscoAddrEvFilter

Interface	Method	Description
boolean	getCiscoAddrParkStatusEvFilter ()	Application can invoke this API to know status of the filter for CiscoAddrParkStatusEv. Default value returned is false.
boolean	getCiscoAddrIntercomInfoChangedEvFilter ()	Application can invoke this API to know the stutus of the filter for CiscoAddrIntercomInfoChangedEv. Default value is true.
boolean	getCiscoAddrIntercomInfoRestorationFailedEvFilter ()	Application can invoke this API to know the status of the filter for CiscoAddrIntercomInfoRestorationFailedEv. Default value is true.
boolean	getCiscoAddrMonitorTerminatedEvFilter ()	Application can invoke this API to know the status of the filter for CiscoAddrMonitorTerminatedEv. Default value is true.

Interface	Method	Description
boolean	getCiscoAddrRecordingConfigChangedEvFilter ()	Application can invoke this API to get the status of the filter for the CiscoAddrRecordingConfigChangedEv. The default value is true.
boolean	getCiscoAddrVoiceMailPilotChangedEvFilter ()	This method returns true if voice mail pilot changed event filter is turned on else false.
void	setCiscoAddrIntercomInfoChangedEvFilter (boolean filter value)	Application can invoke this API to set the status of the filter for CiscoAddrIntercomInfoChangedEv.
void	setCiscoAddrIntercomInfoRestorationFailedEvFilter (boolean filter value)	Application can invoke this API to set the status of the filter for CiscoAddrIntercomInfoRestorationFailedEv.
void	setCiscoAddrMonitorTerminatedEvFilter (Boolean filterValue)	**Parameter** Boolean
Void	setCiscoAddrParkStatusEvFilter (Boolean filterValue)	Application can invoke this API to set the status of the filter for CiscoAddrParkStatusEv.
void	setCiscoAddrRecordingConfigChangedEvFilter (boolean filter value)	Application can invoke this API to set the value of the filter for CiscoAddrRecordingConfigChangedEv.
void	setCiscoAddrVoiceMailPilotChangedEvFilter (boolean filterValue)	This method enables or disables the address voice mail changed event. When this filter is turned on CiscoAddrVoiceMailPilotChangedEv is delivered to address observer when voice mail configuration is changed.

Sample Code

```
CiscoAddress caddr  =  (CiscoAddress) provider.getAddress("2000");
If ( caddr ! =  null ){
   CiscoAddrEvFilter filter  =  caddr.getFilter();
   filter.setCiscoAddrVoiceMailPilotChangedEvFilter(true);
   caddr.addObserver(myAddrObserver);
}

try {
    JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
    MyProviderObserver providerObserver  =  new MyProviderObserver ();
    provider  =  peer.getProvider ( ipaddress;login = useid;passwd = password );
    if ( provider ! =  null ) {
       provider.addObserver ( providerObserver );
       provInService.waitTrue();
       CiscoAddrEvFilter filter;
       CiscoAddress addr  =  provider.getAddress(S1,  S1p);
       filter.setCiscoAddrMonitoringTerminatedEvFilter(false);
       addr.setFilter(filter);
       System.out.println(" Current filter value is : "+
       addr.getFilter().getCiscoAddrMonitoringTerminatedEvFilter());
    }
```

Inherited Methods

None

Parameters

The set methods take a Boolean value as the parameter.

Value Range

The get methods return a Boolean value (true or false).

Related Documentation

See Constant Field Values, on page 1661.

CiscoAddrInServiceEv

The CiscoAddrInServiceEv indicates that the Address is now IN_SERVICE. With Shared Lines (where the same Address appears on different Terminals), applications may receive multiple CiscoAddressInService events for all the Terminals. Applications can use this interface to find out the Terminal on which the Address (or Shared Line) is going IN_SERVICE.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrInServiceEv extends CiscoAddrEv

Fields

Table 49: Fields in CiscoAddrInService

Interface	Field
Static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 50: Methods in CiscoAddrInService

Interface	Method	Description
getTerminal	CiscoTerminal getTerminal()	Returns the Terminal at which this Address is going IN_SERVICE.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Related Documentation, on page 283.getInServiceAddrTerminals() and Constant Field Values, on page 1661 for more information.

CiscoAddrIntercomInfoChangedEv

The system sends the CiscoAddrIntercomInfoChangedEv event to the application whenever the target DN or intercom target label changes for a CiscoIntercomAddress. The system provides this event to all of the application observers that have been added to the CiscoIntercomAddress.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrIntercomInfoChangedEv extends CiscoAddrEv

Fields

Table 51: Fields in CiscoAddrIntercomInfoChangedEv

Interface	Field
Static Int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN,

META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 52: Methods in CiscoAddrIntercomInfoChangedEv

Interface	Method	Description
getIntercomAddress	getIntercomAddress()	Returns the intercom address for which the information changed.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoAddrEv and Constant Field Values, on page 1661 for more information.

CiscoAddrIntercomInfoRestorationFailedEv

The system sends the CiscoAddrIntercomInfoRestorationFailedEv event to the application when JTAPI cannot restore the application set intercom target DN or the intercom target label for the CiscoIntercomAddress during failover or fallback. The system provides this event on the application observer for the application that set the intercom target DN or the intercom target label.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrIntercomInfoRestorationFailedEv extends CiscoAddrEv

Fields

Table 53: Fields in CiscoAddrIntercomInfoRestorationFailedEv

Interface	Field
Static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 54: Methods in CiscoAddrIntercomInfoRestorationFailedEv

Interface	Method	Description
CiscoIntercomAddress	getIntercomAddress()	This interface returns the Cisco IntercomAddress for which intercom information restoration failed.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and CiscoAddrEv for additional information.

CiscoAddrPickupGroupChangedEv

CiscoAddrPickupGroupChangedEv is a new interface being added with Call Pickup feature development. This event is fired whenever a pickup group's information changes, and the line info gets updated. The line info will only be updated when the line is updated with the "apply config" button in the CUCM.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New interface

Declaration

public interface CiscoAddrPickupGroupChangedEv extends CiscoProvEv

Methods

Table 55: Methods in CiscoAddrPickupGroupChangedEv

Interface	Method	Description
CiscoPickupGroup	getOldPickupGroup()	This method returns the old Pickup Group information for this event.
CiscoPickupGroup	getNewPickupGroup()	This method returns the new Pickup Group information for this event, what the pickup group has changed to.

New Error Code

CTIERR_PICKUPGROUP_CHANGED

CTIERR_PICKUPGROUP_DELETED

CiscoAddrOutOfServiceEv

The CiscoAddrOutOfServiceEv event notifies applications that an Address has gone OUT_OF_SERVICE. With Shared Lines(where the same Address appears on different Terminals), applications may receive multiple CiscoAddrOutOfServiceEv events for all the Terminals. Applications can use this interface to find out the Terminal on which the Address(or Shared Line) is going OUT_OF_SERVICE.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, CiscoOutOfServiceEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrOutOfServiceEv extends CiscoAddrEv, CiscoOutOfServiceEv

Fields

Table 56: Fields in CiscoAddrOutOfServiceEv

Interface	Field
Static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface com.cisco.jtapi.extensions.CiscoOutOfServiceEv

CAUSE_CALLMANAGER_FAILURE, CAUSE_CTIMANAGER_FAILURE, CAUSE_DEVICE_FAILURE, CAUSE_DEVICE_RESTRICTED, CAUSE_DEVICE_UNREGISTERED, CAUSE_LINE_RESTRICTED, CAUSE_NOCALLMANAGER_AVAILABLE, CAUSE_REHOME_TO_HIGHER_PRIORITY_CM, CAUSE_REHOMING_FAILURE

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 57: Methods in CiscoAddrOutOfServiceEv

Interface	Method	Description
CiscoTerminal	getTerminal()	Returns the Terminal at which this Address is going OUT_OF_SERVICE.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and Related Documentation, on page 283.getInServiceAddrTerminals() for more information.

CiscoAddrParkStatusEv

When parking a call using the Cisco Unified IP Phone, JTAPI reports park states by using this event. It is provided to all the applications, which have address observers added on the address, which has invoked park. This event gets delivered only when park gets invoked from Cisco Unified IP Phones.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added interface for Park Monitoring and Assisted DPark feature.

Declaration

public interface CiscoAddrParkStatusEv extends CiscoAddrEv

Fields

Table 58: Fields in CiscoAddrParkStatusEv

Interface	Field	Description
static int	PARKED	Park status when the call is parked.
static int	REMINDER	Park status when the park monitoring reversion timer expires.
static int	RETRIEVED	Park status when the parked call is retrieved by the parker or a third party.
static int	FORWARDED	Park status when the parked call is forwarded when the park monitoring forward- no-retrieve timer expires.
static int	ABANDONED	Park status when the parked call is disconnected.

Inherited Fields

None

Methods

Table 59: Methods in CiscoAddrParkStatusEv

Interface	Method	Description
int	getParkState()	Returns the current park state of the parked call.
int	getTransactionID()	Returns an id which is unique for a particular parked call. Transaction ID would remain the same in the different park states for the same parked call.
CiscoCallID	getCiscoCallID()	Returns CiscoCallID.
String	getParkDN()	Returns the DN where call is parked.
String	getParkDNPartition()	Returns the partition of the Park DN.
String	getParkedParty()	Returns the DN of the parked party.
String	getParkedPartyPartition()	Returns the partition of the Parked party.
Terminal	getTerminal()	Returns the terminal on whose address this event is delivered.

Value Ranges

The following are values of fields:

- PARKED: 2
- REMINDER: 3
- RETRIEVED: 4
- ABANDONED: 5
- FORWARDED: 6

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrRecordingConfigChangedEv

The system delivers the CiscoAddrRecordingConfigChangedEv event to the Address Observer if the recording setting on the Address changes.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoAddrRecordingConfigChangedEv extends CiscoAddrEv

Fields

Table 60: Fields in CiscoAddrRecordingConfigChangedEv

Interface	Field
static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 61: Methods in CiscoAddrRecordingConfigChangedEv

Interface	Method	Description
Int	getRecordingConfig()	Returns the new recording configuration on this Address. The value is one of the following: • CiscoAddress.NO_RECORDING • CiscoAddress.AUTO_RECORDING • CiscoAddress.APPLICATION_CONTROLLED_RECORDING
javax.telephony.Terminal	getTerminal()	Returns the Terminal on which the recording configuration changed.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.AddrEv

getAddress

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and CiscoAddrEv for more information.

CiscoAddrRemovedEv

JTAPI sends the CiscoAddrRemovedEv event when an Address gets removed from the Provider domain.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrRemovedEv extends CiscoProvEv

Fields

Table 62: Fields in CiscoAddrRemovedEv

Interface	Field
static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 63: Methods in CiscoAddrRemovedEv

Field	Method	Description
javax.telephony.Address	getAddress()	Returns the Address that is removed from provider domain and the address which is removed from the user control list.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrRemovedFromTerminalEv

The system sends the CiscoAddrRemovedFromTerminalEv event under the following conditions:

- When an Administrator removes a Terminal from the user control list that contains a Shared Line.
- When an Extension Mobility (EM) user logs out from a Terminal with a profile that contains a Shared Line, this event notifies that one of the Terminals got removed from an existing Address.
- When a Shared Line is removed from a Terminal that is in a user control list.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrRemovedFromTerminalEv extends CiscoProvEv

Fields

Table 64: Fields in CiscoAddrRemovedFromTerminalEv

Interface	Field
Static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 65: Methods in CiscoAddrRemovedFromTerminalEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the Address from which the Terminal got removed.
javax.telephony.Terminal	getTerminal()	Returns the Terminal that got removed from the Address.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See also Constant Field Values, on page 1661 for more information.

CiscoAddrRestrictedEv

If an Address is observed and the restriction status is changed to restricted, the system sends this event to the application.

Applications will see this event whenever an Address or an associated Terminal is Restricted from Cisco Unified Communications Manager Administration. For restricted lines, the Address will go OUT_OF_SERVICE and will not come back IN_SERVICE until it is activated again. If an Address is restricted, addCallObserver and addObserver will throw an exception.

For shared lines, if a few shared lines are restricted, and others are not, the system does not throw an exception but the restricted shared lines will not receive any events. If all shared lines are restricted, an exception is thrown when application try adding observers. If an Address gets restricted after observers are added, applications will see CiscoAddrOutOfServiceEv, and when the Address is activated, the Address will go IN_SERVICE.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, CiscoRestrictedEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrRestrictedEv extends CiscoRestrictedEv

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoRestrictedEv

CAUSE_UNKNOWN, CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION, CAUSE_UNSUPPORTED_PROTOCOL, CAUSE_USER_RESTRICTED

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 66: Methods in CiscoAddrRestrictedEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the Address which is changed to Restricted on Cisco Unified Communications Manager.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrRestrictedOnTerminalEv

If the user has Shared lines in the control list, and one of those lines is marked restricted on Cisco Unified Communications Manager, the system sends this event.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, CiscoRestrictedEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoAddrRestrictedOnTerminalEv extends CiscoRestrictedEv

Fields

Table 67: Fields in CiscoAddrRestrictedOnTerminalEv

Interface	Field
Static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoRestrictedEv

CAUSE_UNKNOWN, CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION, CAUSE_UNSUPPORTED_PROTOCOL, CAUSE_USER_RESTRICTED

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS,

META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 68: Methods in CiscoAddrRestricedOnTerminalEv

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the Address that is restricted.
javax.telephony.Terminal	getTerminal()	Returns the Terminal on which the Address is restricted.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAddrVoiceMailPilotChangedEv

This event indicates that the voice mail pilot configuration on address is changed and is delivered to address observer. Application needs to enable the corresponding filter in CiscoAddrEvFilter to receive this event.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(3)	New interface.

Sample Code:

```
class myAddrObserver  extends CiscoAddressObserver {
  public synchronized void addressChangedEvent ( AddrEv [] eventList ) {
    if ( eventList[i] instanceof CiscoAddrVoiceMailPilotChangedEv){
       CiscoAddrVoiceMailPilotChangedEv ev  =  (CiscoAddrVoiceMailPilotChangedEv)

                                      eventList[i];
       Address cAddr  =  ev.getAddress();
       String newVoiceMailPilot  =  ev.getVoiceMailPilot();
       System.out.println(" New voice mail pilot for " +
         ev.getAddress()  +   " is " +  newVoiceMailPilot );
    }
  }
}
```

Superinterfaces

NA

Declaration

NA

Fields

Table 69: Fields in CiscoAddrVoiceMailPilotChangedEv

Interface	Field

Inherited Fields

Methods

Table 70: Methods in CiscoAddrVoiceMailPilotChangedEv

Interface	Method	Description
String	getVoiceMailPilot()	This method returns the new voice mail pilot of the address.

Inherited Methods

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoAnnouncementStartedEv

CiscoAnnouncementStartedEv is a new JTAPI event that is delivered to applications as a Call Event. This new event is delivered to call observers added by applications to notify when a play announcement starts.

Interface History

Cisco Unified Communications Manager Release Number	Description
10.01	Created history table to track changes

Declaration

Public interface CiscoAnnouncementStartedEv extends CiscoCallEv.

Methods

Interface	Method	Description
String	getAnnouncementID ()	This interface returns the name of the announcement identifier.

CiscoAnnouncementEndedEv

CiscoAnnouncementEndedEv is a new JTAPI event that is delivered to applications as a Call Event. This new event is delivered to call observers added by applications to notify when play announcement ends.

Interface History

Cisco Unified Communications Manager Release Number	Description
10.01	Created history table to track changes

Declaration

Public interface CiscoAnnouncementEndedEv extends CiscoCallEv.

Methods

Interface	Method	Description
boolean	getSuccess()	This interface returns whether or not the play announcement was successful. Returns true if there are no errors with the play announcement, or returns false indicating error.
int	getErrorCode()	This interface returns the error code indicating the cause of the failure/error with play announcment. This maps to one of the values defined in CiscoJtapiException.
String	getErrorDescription()	This interface returns the string corresponding to what the error code maps to.

CiscoAnnouncementErrorEv

CiscoAnnouncementErrorEv is a new JTAPI event that is delivered to applications as a Call Event. This new event is delivered to call observers added by applications to notify when an error occurs during play announcement.

Interface History

Cisco Unified Communications Manager Release Number	Description
10.01	Created history table to track changes

Declaration

Public interface CiscoAnnouncementErrorEv extends CiscoCallEv.

Methods

Interface	Method	Description
Int	getErrorCode()	This interface returns the error code indicating the cause for the failure/error with play announcment. This maps to one of the values defined in CiscoJtapiException.
String	getErrorDescription()	This interface returns the string corresponding to what the error code maps to.

CiscoBaseMediaTerminal

The CiscoBaseMediaTerminal interface extends the CiscoTerminal interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.5(1)	New interface.

Declaration

public interface CiscoBaseMediaTerminal extends CiscoTerminal

Superinterfaces

NA

Fields

Table 71: Fields in CiscoBaseMediaTerminal

Interface	Field
Final static int	NO_MEDIA_REGISTRATION
Final static int	DYNAMIC_MEDIA_REGISTRATION
Final static int	DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT
Final static int	STATIC_MEDIA_REGISTRATION
Final static int	STATIC_MEDIA_REGISTRATION_FOR_GET_PORT SUPPORT

Inherited Fields

NA

Methods

Table 72: Methods in CiscoBaseMediaTerminal

Interface	Method
int	getRegistrationType()
void	register(java.net.InetAddress address, int port, CiscoMediaCapability[] capabilities, int registrationType), int[] algorithmIDs, java.net.InetAddress address_v6, int activeAddressingMode) throws CiscoRegistrationException, PrivilegeViolationException;

Inherited Methods

NA

Parameters

- register()
- Java.net.InteAddress address
- int port
- CiscoMediaCapability[] capabilities
- int[] algorithmIDs
- Java.net.InteAddress address_v6
- int activeAddressingMode
- int registrationType

Data Types

- Java.net.InteAddress address
- int port
- CiscoMediaCapability[] capabilities
- int[] algorithmIDs
- Java.net.InteAddress address v6
- int activeAddressingMode
- int registrationType

Range of Values

- activeAddressingMode:
  - CiscoTerminal.IP_ADDRESSING_MODE_IPv4 or
  - CiscoTerminal.IP_ADDRESSING_MODE_IPv6 or
  - CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6
- registrationType:
  - CiscoTerminal.NO_MEDIA_REGISTATION (applicable only for route points)
  - CiscoTerminal.DYNAMIC_MEDIA_REGISTRATION
  - CiscoTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT
  - CiscoTerminal.STATIC_MEDIA_REGISTRATION

• CiscoTerminal.STATIC_MEIDA_TERMINAL_FOR_GET_PORT_SUPPORT

CiscoCall

The CiscoCall interface extends the CallControlCall interface with additional Cisco Unified Communications Manager specific capabilities.

In Cisco Unified Communications Manager, every Call object comprises a set of call legs that share a common identifier: the global call handle. Connection objects represent call legs in JTAPI, and the Call object that relates a set of connections contains the global call handle that the underlying call legs share.

The global call handle within a CiscoCall is accessible by using CallManagerID and CallID properties. Taken together, the CallManagerID and CallID form the global call handle that Cisco Unified Communications Manager maintains. Consider this pair of properties as guaranteed to be unique among all ACTIVE Call objects, but when an ACTIVE call becomes INACTIVE, its CallManagerID and CallID may be reused to identify a newly created Call object. Therefore, an INACTIVE Call can have identical CallManagerID and CallID properties to those of a currently ACTIVE Call object.

Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	Two new APIs: CiscoMultiMediaCapabilityInfogetCallingTerminalMultiMediaCapabilityInfo() Returns the calling party terminal multimedia capability. CiscoMultiMediaCapabilityInfogetCalledTerminalMultiMediaCapabilityInfo() Returns the called party terminal multimedia capability. Three new constants: CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_NONE CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_G
9.0(1)	Five new constants: CALL_RECORDING_TYPE_NONE, CALL_RECORDING_TYPE_AUTOMATIC, CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT, CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION, and CALL_RECORDING_TYPE_USER_INITIATED_FROM_DEVICE, are added.
8.0(1)	Enhanced with the following: New methods that allow applications to get the Terminals associated with the current calling and current called parties on the call. New API to indicate whether the call is created due to CallFwdAll key press or not. Three new constants, CFWD_ALL_NONE, CFWD_ALL_SET, and CFWD_ALL_CLEAR, have been introduced for CiscoCall interface.

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Added new method called isConference() for Drop Any Party feature.

Superinterfaces

javax.telephony.Call, javax.telephony.callcontrol.CallControlCall, CiscoObjectContainer

Subinterfaces

CiscoConsultCall

Declaration

public interface CiscoCall extends javax.telephony.callcontrol.CallControlCall, CiscoObjectContainer

Fields

Table 73: Fields in CiscoCall

Interface	Field	Description
static int	CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT	This constant is used when silent is the recording invocation type. Silent recording is the default value in releases prior to Release 9.0.
static int	CALL_RECORDING_TYPE_AUTOMATIC	This constant is used when recording is invoked automatically by Unified CM, as a result of the line configuration.
static int	CALL_RECORDING_TYPE_NONE	This constant is used when a call is not recorded.
static int	CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION	This constant is used when user is the recording invocation type, and the request was invoked by an application.
static int	CALL_RECORDING_TYPE_USER_INITIATED_FROM_DEVICE	This constant is used when recording was invoked on the Cisco IP device.
Static int	CALLSECURITY_AUTHENTICATED	Call security status is authenticated.
Static int	CALLSECURITY_ENCRYPTED	Call security status is encrypted.
Static int	CALLSECURITY_NOTAUTHENTICATED	Call security status is not authenticated.

Interface	Field	Description
Static int	CALLSECURITY_UNKNOWN	Call security status is unknown.
public static final int	CFWD_ALL_NONE	When call is not created due to CallFwdAll soft key press. Value is 0.
public static final int	CFWD_ALL_SET	When call is created due to CallFwdAll key press to set CFA. Value is 64.
public static final int	CFWD_ALL_CLEAR	When call is created to CallFwdAll key press to clear/cancel CFA. Value is 128.
Static int	FEATUREPRIORITY_EMERGENCY	Feature priority is emergency
Static int	FEATUREPRIORITY_NORMAL	Feature priority is normal
Static int	FEATUREPRIORITY_URGENT	Feature priority is urgent
int	getCFwdAllKeyPressIndicator()	This interface indicates if call is created due to callFWDAll Key press. It returns one of the following: • CiscoCall.CFWD_ALL_NONE • CiscoCall.CFWD_ALL_SET • CiscoCall.CFWD_ALL_CLEAR
Static int	PLAYTONE_BOTHLOCALANDREMOTE	A tone plays to both the caller and the monitor target (agent) when this option gets used.
Static int	PLAYTONE_LOCALONLY	A tone plays only to the monitor target (agent) when this option gets used.
Static int	PLAYTONE_NOLOCAL_OR_REMOTE	When this option is used no tone plays to the monitor target (agent) or the caller.
Static int	PLAYTONE_REMOTEONLY	A tone plays only to the caller when this option gets used.
Static int	SILENT_MONITOR	This option indicates that silent monitor is requested.
static final int	CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_NONE	This option indicates that there is no Media Forking Device for recording on this call. Range of value = 0

Interface	Field	Description
static final int	CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE	This option indicates that the Media Forking Device type for recording on this call is Phone (BIB Recording). Range of value = 1
static final int	CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW	This option indicates that the Media Forking Device type for recording on this call is Gateway (GW Recording). Range of value = 2

Inherited Fields

From Interface javax.telephony.Call

ACTIVE, IDLE, INVALID

Sample Code

```
public class MyCallObserver implements implements CallObserver,
           CallControlCallObserver,  MediaCallObserver {
  public void callChangedEvent (CallEv[] evlist) {
    for(int i = 0; evlist ! =  null && i < evlist.length; i++){
…
…
      If (evlisth[i] instance of TermConnActiveEv){
         CiscoCall thisCall  =  (CiscoCall) ((TermConnActiveEv)
evlist[i]).getCall();
         int cfaStatus  =  thisCall.getCFWDAllKeyPressIndicator();
         If (cfaStatus  =  =  CiscoCall.CFWD_ALL_SET ||
             cfaStatus  =  =  CiscoCall.CFWD_ALL_CLEAR){
          System.out.println("Call is created due to CallFwdAll soft key press");

         }else {
            System.out.println("Call is NOT created due to CallFwdAll soft key
press");
      }
    }
  }
        …
        …
}
```

Methods

Table 74: Methods in CiscoCall

Interface	Method	Description
Void	conference (javax. telephony. Call[]otherCalls)	This interface conferences multiple calls together, resulting in the union of the participants of all the calls being placed on a single call.
java. lang. boolean	isConference ()	Returns True if it is a conference call, false or otherwise.
javax. telephony. Connection[]	connect (javax. telephony. Terminal origterm, javax. telephony. Addressorigaddr java. lang. String. dialedDigits int featurePriority)	This method overloads Call. connect (). It takes a new parameter featurePriority. The featurePriority parameter may be: CiscoCall. FEATUREPRIORITY_NORMAL CiscoCall. FEATUREPRIORITY_URGENT CiscoCall. FEATUREPRIORITY_EMERGENCY Throws: javax. telephony. ResourceUnavailableException, javax. telephony. PrivilegeViolationException, javax. telephony. InvalidPartyException, javax. telephony. InvalidArgumentException, javax. telephony. InvalidStateException, javax. telephony. MethodNotSupportedException
boolean	getCalledAddressPI ()	Returns the Presentation Indicator (PI) that is associated with getCalledAddressPI.
CiscoPartyInfo	getCalledPartyInfo ()	Returns the PartyInfo of the called party of the call.
CiscoCallID	getCallID ()	CallID is a unique identifier among all ACTIVE calls with the same CallManagerID.
boolean	getCallingAddressPI ()	Returns the Presentation Indicator (PI) that is associated with getCallingAddressPI.
int	getCallSecurityStatus ()	This interface returns the SecurityStatus of the Call.
CiscoConference Chain	getConferenceChain ()	This interface returns a CiscoConferenceChain object if this Call is a chained conference Call.
javax. telephony. Address	getCurrentCalledAddress ()	Returns the current called Address for the call.
boolean	getCurrentCalledAddressPI ()	Returns the Presentation Indicator (PI) that is associated with CurrentCalledAddress.
boolean	getCurrentCalledDisplayNamePI ()	Returns the Presentation Indicator (PI) that is associated with getCurredCalledDisplayNamePI.

Interface	Method	Description
java. lang. String	getCurrentCalledPartyDisplayName ()	This interface returns the display name of the called party in the call.
CiscoPartyInfo	getCurrentCalledPartyInfo ()	Returns the PartyInfo of the current called party of the call.
java. lang. String	getCurrentCalledPartyUnicodeDisplayName ()	Returns the Unicode display name of the called party in the call.
int	getCurrentCalledPartyUnicodeDisplayNamelocale ()	Returns the locale of the current called party Unicode display name.
javax. telephony. Address	getCurrentCallingAddress ()	Returns the current calling Address for the call.
boolean	getCurrentCallingAddressPI ()	Returns the Presentation Indicator (PI) that is associated with getCurrentCallingAddressPI.
boolean	getCurrentCallingDisplayNamePI ()	Returns the Presentation Indicator (PI) that is associated with getCurrentCalledDisplayNamePI.
java. lang. String	getCurrentCallingPartyDisplayName ()	This interface returns the display name of the calling party.
CiscoPartyInfo	getCurrentCallingPartyInfo ()	Returns the PartyInfo of the current calling party of the call.
java. lang. String	getCurrentCallingPartyUnicodeDisplayName ()	Returns the Unicode display name of the calling party in the call.
int	getCurrentCallingPartyUnicodeDisplayNamelocale ()	Returns the locale of the current called party Unicode display name.
Terminal	getCurrentCallingTerminal ()	This method returns a Terminal object that represents the terminal of the calling party on the call. By default, if the terminal is not defined, these will return null. An example of when this would occur is when a phoen goes offhook, and a one-sided call is created. The CalledTerminal would be null in this scenario. The terminal for the called party is only set AFTER the called party answers a call.
Terminal	getCurrentCalledTerminal ()	This method returns a Terminal object that represents the terminal of the called party on the call. By default, if the terminal is not defined, these will return null. An example of when this would occur is when a phoen goes offhook, and a one-sided call is created. The CalledTerminal would be null in this scenario. The terminal for the called party is only set AFTER the called party answers a call.

Interface	Method	Description
java. lang. String	getGlobalizedCallingParty ()	This will return the globalizedCallingParty
CiscoPartyInfo	getLastRedirectedPartyInfo ()	Returns the PartyInfo of the last redirecting party of the call.
boolean	getLastRedirectingAddressPI ()	Returns the Presentation Indicator (PI) that is associated with getLastRedirectingAddressPI.
CiscoPartyInfo	getLastRedirectingPartyInfo ()	Deprecated. - use getLastRedirectedPartyInfo ();
javax. telephony. Address	getModifiedCalledAddress ()	This interface returns the modified called Address for the call if called party is modified by using called party transformation pattern or other means.
javax. telephony. Address	getModifiedCallingAddress ()	This interface returns the modified calling Address for the call if an application modifies its calling party by using the selectRoute API or other means.
boolean	isPersistentCall ()	This interface returns true if the call is a persistent call and false if the call is a normal call.
javax. telephony. Connection[]	startMonitor (javax. telephony. TerminalMonitorInitiatorterminal, javax. telephony. AddressMonitorInitiatoraddress, intmonitorTargetcallid, java. lang. StringmonitorTargetDN, java. lang. StringmonitorTargetTerminalName, intmonitorType, intplayToneDirection)	If the application has the information about the call at the monitor target, the application can use this interface to monitor calls.
javax. telephony. Connection[]	startMonitor (javax. telephony. TerminalMonitorInitiatorterminal, javax. telephony. AddressMonitorInitiatoraddress, javax. telephony. TerminalConnectiontermConnofMonitorTarget, intmonitorType, intPlayToneDirection)	If the application is observing the monitor target (agent) Address, the application can use the Terminal connection of the monitor target (agent) to initiate a monitor request.
javax. telephony. Connection	transfer (java. lang. Stringaddress, java. lang. StringfacCode, java. lang. StringcmcCode)	This method is similar to the CallControlCall. transfer (String address) interface except that it also takes facCode (Forced Authorization Code) and cmcCode (Client Matter Code) if the transfer Address requires these codes to offer the call.
CiscoMultiMedia CapabilityInfo	getCallingTerminalMultiMediaCapabilityInfo ()	Returns the calling party terminal multimedia capability.
CiscoMultiMedia CapabilityInfo	getCalledTerminalMultiMediaCapabilityInfo ()	Returns the called party terminal multimedia capability.

Note In Cisco Unified JTAPI implementation, CallControlCall.getCalledAddress() returns the first called party of the call which is the original called party.

Inherited Methods

From Interface javax.telephony.callcontrol.CallControlCall

addParty, conference, consult, consult, drop, getCalledAddress, getCallingAddress, getCallingTerminal, getConferenceController, getConferenceEnable, getLastRedirectedAddress, getTransferController, getTransferEnable, offHook, setConferenceController, setConferenceEnable, setTransferController, setTransferEnable, transfer, transfer

From Interface javax.telephony.Call

addObserver, connect, getCallCapabilities, getCapabilities, getConnections, getObservers, getProvider, getState, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Parameters

- origterm -
- origaddr -
- dialedDigits -
- featurePriority -

Conference Controller

For the conferencing feature to happen, a common participant must belong to all the Calls, as represented TerminalConnection of common participants on controller Terminal. These TerminalConnections are known as the conference controllers. At the most, only one of TerminalConnection on the Calls at controller Terminal would be in CallControlTerminalConnection.TALKING state, and hence, the TerminalConnection on the secondary Call should be in the CallControlTerminalConnection.HELD state. As a result of invokation of this method, all the conference controller TerminalConnection merge into one TerminalConnection.

Applications can set which Terminal would acts as the conference controller when a conference call gets set up by setting up Conference controller TerminalConnection via invoking CallControlCalll.setConferenceController() method. The CalControlCall.getConferenceController() method returns the current conference controller, or null if there is none. If no conference controller is set initially, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked.

Telephone Call Argument

All participants from the secondary Calls, passed as the argument to this method, move to the Call on which this method was invoked. That is, new Connections and TerminalConnections for the participant in the secondary Calls are created on this Call. The Connections and TerminalConnections on the secondary Calls get removed from the Call, and the Call moves to the Call.INVALID state.

Other Shared Participants

There may exist other Addresses and Terminals that are part of some calls in addition to the designated conference controller. In these instances, those participants that are shared between both Calls are merged into one. That is, the Connections and TerminalConnections on this Calls stay unchanged. The corresponding Connections and TerminalConnections on the secondary Calls get removed from that Call.

Pre-Conditions

1. Let tc1 be the conference controller on this Call
2. Let connection1 = tc1.getConnection()
3. Let tc2 to tcN be the conference controllers on otherCalls
4. (this.getProvider()).getState() = = Provider.IN_SERVICE
5. this.getState() = = Call.ACTIVE
6. tc1.getTerminal() = = tc2.getTerminal()... = tcN.getTerminal
7. tc1.getCallControlState() = = CallControlTerminalConnection.TALKING/HELD
8. tc2-tcN.getCallControlState() = = CallControlTerminalConnection.HELD/TALKING
9. this ! = otherCalls

Post-Conditions

1. (this.getProvider()).getState() = = Provider.IN_SERVICE
2. this.getState() = = Call.ACTIVE
3. otherCall.getState() = = INVALID
4. Let c[] be the Connections to be merged from otherCall
5. Let tc[] be the TerminalConnections to be merged from otherCall
6. Let new(c) be the set of new Connections created on this Call
7. Let new(tc) be the set of new TerminalConnections created on this Call
8. new(c) element of this.getConnections()
9. new(c).getCallState() = = c.getCallState()
10. new(tc) element of (this.getConnections()).getTerminalConnections()
11. new(tc).getCallState() = = tc.getCallState()
12. c[i].getCallControlState() = = CallControlConnection.DISCONNECTED for all i
13. tc[i].getCallControlState() = = CallControlTerminalConnection.DROPPED for all i
14. CallInvalidEv is delivered for otherCall
15. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for all c[i]
16. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for all tc[i]

17. ConnCreatedEv is delivered for all new(c)
18. TermConnCreatedEv is delivered for all new(tc)
19. Appropriate events are delivered for all new(c) and new(tc)

Parameters

otherCalls - The Other Calls which are to be merged with this Call object.

Throws

javax.telephony.InvalidArgumentException - The Call object that is provided is not valid for the conference.

javax.telephony.InvalidStateException - Thismeans that the Provider is not "in service, " the Call is not "active, " or the conference controllers are not in the proper state.

javax.telephony.MethodNotSupportedException - The implementation does not support this method.

javax.telephony.PrivilegeViolationException - The application does not have the proper authority to invoke this method.

javax.telephony.ResourceUnavailableException - This means that an internal resource that is necessary for the successful invocation of this method is not available.

See Also

ConnCreatedEv, TermConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

javax.telephony.Connection transfer(java.lang.Stringaddress java.lang.StringfacCode, java.lang.StringcmcCode)

Throws for connect(Terminal, Address, String, CiscoRTPParams)

javax.telephony.InvalidArgumentException, javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupportedException, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailableExceptionThis method is similar to the CallControlCall.transfer(String address) interface except that it also takes facCode (Forced Authorization Code) and cmcCode (Client Matter Code) if the transfer Address requires these codes to offer the call. If only one of the codes is required, the other code may need to be a null value.

If the user enters no codes, or invalid codes, the call may not be offered and platformException may contain the following error codes:

CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_INVALID
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_INVALID

This overloaded version of this method transfers all participants currently on this Call, with the exception of the transfer controller participant, to another Address. This is often called a "single-step transfer" because the transfer feature places another call and performs the transfer simultaneously. The Address string argument to this method must be valid and complete.

The Transfer Controller

The transfer controller for this version of this method represents the participant on this Call around which the transfer is taking place and who drops off the Call after the transfer has completed. The transfer controller is a TerminalConnection that must be in the CallControlTerminalConnection.TALKING state.

Applications may control which TerminalConnection acts as the transfer controller via the CallControlCall.setTransferController() method. The CallControlCall.getTransferController() method returns the current transfer controller, or null if there is none. If no transfer controller is set, the implementation chooses a suitable TerminalConnection when the transfer feature gets invoked.

When the transfer feature gets invoked, the transfer controller moves into the CallControlTerminalConnection.DROPPED state. If it is the only TerminalConnection associated with its Connection, then its Connection moves into the CallControlConnection.DISCONNECTED state as well.

The New Connection

This method creates and returns a new Connection representing the party to which the Call was transferred. This Connection may be null if the Call has been transferred outside of the Provider domain and can no longer be tracked. This Connection must at least be in the CallControlConnection.IDLE state. The Connection state may have progressed beyond "idle" before this method returns, and should be reflected by an event. This new Connection will progress as any normal destination Connection on a call. Typical scenarios for this Connection are described by the Call.connect() method.

Pre-Conditions

1. Let tc be the transfer controller on this Call
2. (this.getProvider()).getState() = = Provider.IN_SERVICE
3. this.getState() = = Call.ACTIVE
4. tc.getCallControlState() = = CallControlTerminalConnection.TALKING

Post-Conditions

1. Let newconnection be the Connection created and returned
2. Let connection = = tc.getConnection()
3. (this.getProvider()).getState() = = Provider.IN_SERVICE
4. this.getState() = = Call.ACTIVE
5. tc.getCallControlState() = = CallControlTerminalConnection.DROPPED
6. If connection.getTerminalConnections().length = = 1, then connection.getCallControlState() = = CallControlConnection.DISCONNECTED
7. newconnection is an element of this.getConnections(), if not null.
8. newconnection.getCallControlState() at least CallControlConnection.IDLE, if not null.
9. ConnCreatedEv is delivered for newconnection
10. CallCtlTermConnDroppedEv/TermConnDroppedEv is delivered for tc

11. CallCtlConnDisconnectedEv/ConnDisconnectedEv is delivered for connection if no other TerminalConnections

Parameters

- address - The destination Address string(dialedDigits) to which the Call is being transferred.
- facCode - The Force Authorization Code
- cmcCode - The Client Matter Code

Returns

The new Connection associated with the destination, or null.

Throws

javax.telephony.InvalidArgumentException - The TerminalConnection provided as controlling the transfer is not valid or not part of this Call.

javax.telephony.InvalidStateException - This means that the Provider is not "in service, " the Call is not "active, " or the transfer controller is not "talking."

javax.telephony.InvalidPartyException - The destination Address is not valid or complete.

javax.telephony.MethodNotSupportedException - The implementation does not support this method.

javax.telephony.PrivilegeViolationException - The application does not have the proper authority to invoke this method.

javax.telephony.ResourceUnavailableException - An internal resource necessary for the successful invocation of this method is unavailable.

See Also

ConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

getCurrentCalledAddressPIboolean getCurrentCalledAddressPI()Returns the Presentation Indicator(PI) that is associated with CurrentCalledAddress. If it returns true, the application can display this Address name to the end users. If it returns false, the application should not display this Address name to end users.

getCurrentCalledDisplayNamePIboolean getCurrentCalledDisplayNamePI()Returns the Presentation Indicator(PI) that is associated with getCurredCalledDisplayNamePI. If it returns true, the application can display this DisplayName to the end users. If it returns false, the application should not display this DisplayName to the end users.

getCurrentCallingAddressPIboolean getCurrentCallingAddressPI()Returns the Presentation Indicator(PI) that is associated with getCurrentCallingAddressPI. If it returns true, the application can display this Address name to the end users. If it returns false, the application should not display this Address name to the end users.

getCurrentCallingDisplayNamePIboolean getCurrentCallingDisplayNamePI()Returns the Presentation Indicator(PI) that is associated with getCurrentCalledDisplayNamePI. If it returns true, the application can display this DisplayName to the end users. If it returns false, the application should not display this DisplayName to the end users.

getLastRedirectingAddressPIboolean getLastRedirectingAddressPI()Returns the Presentation Indicator(PI) that is associated with getLastRedirectingAddressPI. If it returns true, the application can display this Address name to the end users. If it returns false, the application should not display this Address name to the end users.

getCalledAddressPIboolean getCalledAddressPI()Returns the Presentation Indicator(PI) that is associated with getCalledAddressPI. If it returns true, the application can display this Address name to the end users If it returns false, the application should not display this Address name to the end users.

getCallingAddressPIboolean getCallingAddressPI()Returns the Presentation Indicator(PI) that is associated with getCallingAddressPI. If it returns true, the application can display this Address name to the end users. If it returns false, the application should not display this Address name to the end users.

getCurrentCalledPartyUnicodeDisplayNamejava.lang.String getCurrentCalledPartyUnicodeDisplayName()Returns the Unicode display name of the called party in the call. It returns null if the display name is unknown.

getCurrentCalledPartyUnicodeDisplayNamelocaleint getCurrentCalledPartyUnicodeDisplayNamelocale()Returns the locale of the current called party Unicode display name. CiscoLocale interface lists the supported locales.

getCurrentCallingPartyUnicodeDisplayNamejava.lang.String getCurrentCallingPartyUnicodeDisplayName()Returns the Unicode display name of the calling party in the call. It returns null if the display name is unknown.

getCurrentCallingPartyUnicodeDisplayNamelocaleint getCurrentCallingPartyUnicodeDisplayNamelocale()Returns the locale of the current called party Unicode display name.

getCurrentCallingPartyInfoCiscoPartyInfo getCurrentCallingPartyInfo()Returns the PartyInfo of the current calling party of the call.

getCurrentCalledPartyInfoCiscoPartyInfo getCurrentCalledPartyInfo()Returns the PartyInfo of the current called party of the call.

getLastRedirectingPartyInfoCiscoPartyInfo getLastRedirectingPartyInfo()Deprecated.- use getLastRedirectedPartyInfo();

Returns the PartyInfo of the last redirecting party of the call.

getLastRedirectedPartyInfoCiscoPartyInfo getLastRedirectedPartyInfo()Returns the PartyInfo of the last redirecting party of the call.

getCalledPartyInfoCiscoPartyInfo getCalledPartyInfo()Returns the PartyInfo of the called party of the call.

javax.telephony.Connection[]startMonitor(javax.telephony.TerminalMonitorInitiatorterminal, javax.telephony.AddressMonitorInitiatoraddress, javax.telephony.TerminalConnectiontermConnofMonitorTarget, intmonitorType, intPlayToneDirection)

throws

javax.telephony.ResourceUnavailableException, javax.telephony.PrivilegeViolationException, javax.telephony.InvalidPartyException, javax.telephony.InvalidArgumentException, javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException

If the application is observing the monitor target (agent) Address, the application can use the Terminal connection of the monitor target (agent) to initiate a monitor request. This interface places a call from an originating endpoint to monitor the call at the monitor target.

Pre-Conditions

1. (this.getProvider()).getState() = = Provider.IN_SERVICE
2. this.getState() = = Call.IDLE
3. ((CiscoProviderCapabilities)(this.getTerminal().getProvider().getProviderCapabilities()).canMonitor() = = TRUE
4. TerminalConnection.getProvider() = = this.getProvider()

Parameters

- MonitorInitiatorterminal - - The originating Terminal
- MonitorInitiatoraddress - - The originating Address
- termConnofMonitorTarget - - The TerminalConnection of the target
- monitorType - - The type of monitor. Use CiscoCall.SILENT_MONITOR.
- PlayToneDirection - - Indicates whether the tone needs to be played to the target, the initiator, or both. This should be one of CiscoCall.PLAYTONE_NOLOCAL_OR_REMOTE, CiscoCall.PLAYTONE_LOCALONLY, CiscoCall.PLAYTONE_REMOTEONLY, or CiscoCall.PLAYTONE_BOTHLOCALANDREMOTE

Throws

javax.telephony.ResourceUnavailableException

javax.telephony.PrivilegeViolationException

javax.telephony.InvalidPartyException

javax.telephony.InvalidArgumentException

javax.telephony.InvalidStateException

javax.telephony.MethodNotSupportedException

Related Documentation

See CallControlCall for more information.

CiscoCallChangedEv

The system delivers the CiscoCallChangedEv event to the call observer for all supported features whenever the Global Call ID (GCID) of the call changes. CiscoCallChangedEv gets delivered when the GCID of the call changes due to path replacement (QSIG_PR) and for other features, including transfer, conference, barge, cbarge, and unpark. In the case of shared lines, multiple CiscoCallChangedEv events get delivered.

The system also delivers this event when two or more calls get merged into one. Transfer, conference, unpark, Barge, and CBarge will trigger this event. Application can invoke CiscoCallEv.getCiscoFeatureReason() to find the feature code that caused this event.

The system reports this event via the CallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoCallChangedEv extends CiscoCallEv

Fields

Table 75: Fields in CiscoCallChangedEv

Interface	Field
static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,

CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 76: Methods in CiscoCallChangedEv

Interface	Method	Description
Interface	getConnection()	Returns the CiscoConnection to the Address where the change occurred.
CiscoConnection	getOriginalCall()	Returns the call that will go to INVALID state.
CiscoCall	getSurvivingCall()	Returns the call that will remain active after the callID change.
CiscoCall	getTerminalConnection()	Returns the TerminalConnection where the change occurred. This value could be null if the call ID changes before the TerminalConnection gets created on the Address.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoCallConsultCancelledEv

This event notifies applications that a cancel operation has been invoked.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	New event for the Swap/Cancel - Transfer/Conference Behavior Change feature.

Superinterfaces

None

Declaration

public interface CiscoCallConsultCancelledEv

Fields

None

Inherited Fields

None

Methods

Table 77: Methods in CiscoCallConsultCancelledEv

Interface	Method	Description
CiscoCall	getConsultCall()	Returns the consult call for which consult operation is cancelled. If the consult call does not exist, it returns NULL. The getCall() API on this call event returns the parent call.

Inherited Methods

None

Related Documentation

None.

CiscoCallCtlConnOfferedEv

The CiscoCallCtlConnOfferedEv interface extends the CallCtlConnOfferedEv interface to let applications obtain the IP Address of the calling party Terminal. The IP Address information might not be available for all calling party devices. A return value of 0 (or null) indicates that the information is not available.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlConnEv, javax.telephony.callcontrol.events.CallCtlConnOfferedEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.events.CallEv, javax.telephony.events.ConnEv, javax.telephony.events.Ev

Declaration

public interface CiscoCallCtlConnOfferedEv extends javax.telephony.callcontrol.events.CallCtlConnOfferedEv

Fields

None

Inherited Fields

From Interface javax.telephony.callcontrol.events.CallCtlConnOfferedEv

None

From Interface javax.telephony.callcontrol.events.CallCtlEv

CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION,

CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 78: Methods in CiscoCallCtlConnOfferedEv

Interface	Method	Description
java.net.InetAddress	getCallingPartyIpAddr()	Returns the IP address of the calling party, or 0 (or null) if the IP Address is not available.

Inherited Methods

From Interface javax.telephony.callcontrol.events.CallCtlCallEv

getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

From Interface javax.telephony.callcontrol.events.CallCtlEv

getCallControlCause

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ConnEv

getConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

None

CiscoCallCtlTermConnHeldReversionEv

The CiscoCallCtlTermConnHeldReversionEv event indicates that hold reversion notification has been received on the TerminalConnection from Cisco Unified Communications Manager.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.callcontrol.events.CallCtlTermConnEv, javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoCallCtlTermConnHeldReversionEv extends javax.telephony.callcontrol.events.CallCtlTermConnEv

Fields

Table 79: Fields in CiscoCallCtlTermConnHeldReversionEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.callcontrol.events.CallCtlEv

CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.callcontrol.events.CallCtlCallEv

getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

From Interface javax.telephony.callcontrol.events.CallCtlEv

getCallControlCause

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoCallEv

The CiscoCallEv interface, which extends the JTAPI core javax.telephony.events.CallEv interface, serves as the base interface for all Cisco-extended JTAPI Call events. Every Call-related event in this package extends this interface, directly or indirectly.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoEv, javax.telephony.events.Ev

Subinterfaces

CiscoCallChangedEv, CiscoCallSecurityStatusChangedEv, CiscoConferenceChainAddedEv, CiscoConferenceChainRemovedEv, CiscoConferenceEndEv, CiscoConferenceStartEv, CiscoConsultCallActiveEv, CiscoToneChangedEv, CiscoTransferEndEv, CiscoTransferStartEv

Declaration

public interface CiscoCallEv extends CiscoEv, javax.telephony.events.CallEv

Fields

Table 80: Fields in CiscoCallEv

Interface	Field	Description
Static int	CAUSE_ACCESSINFORMATIONDISCARDED	This cause indicates that the network could not deliver access information to the remote user as requested.
Static int	CAUSE_BARGE	It indicates the call is a BARGE call.
staticint	CAUSE_BCBPRESENTLYAVAIL	This cause indicates that the user has requested a bearer capability which is implemented by the equipment which generated this cause but which is not available at this time.
static int	CAUSE_BCNAUTHORIZED	This cause indicates that the user has requested a bearer capability which is implemented by the equipment which generated this cause but the user is not authorized to use.
static int	CAUSE_BEARERCAPNIMPL	This cause indicates that the equipment sending this cause does not support the bearer capability requested.
static int	CAUSE_CALLBEINGDELIVERED	This cause indicates that the user has been awarded the incoming call and that the incoming call is being connected to a channel already established to that user for similar calls.
static int	CAUSE_CALLIDINUSE	This cause indicates that the network has received a call suspended request containing a call identity (including the null call identity) which is already in use for a suspended call within the domain of interfaces over which the call might be resumed.
static int	CAUSE_CALLMANAGER_FAILURE	This cause indicates the failure due to CALL Manager Failure.

Interface	Field	Description
static int	CAUSE_CALLREJECTED	This cause indicates that the equipment sending this cause does not wish to accept this call.
static int	CAUSE_CALLSPLIT	This cause indicates the call split, it could mean conference or transfer.
static int	CAUSE_CHANTYPENIMPL	This cause indicates that the equipment sending this cause does not support the channel type requested.
static int	CAUSE_CHANUNACCEPTABLE	This cause indicates that the channel most recently identified is not acceptable to the sending entity for use in this call.
static int	CAUSE_CTICCMSIP400BADREQUEST	This cause indicates the call is rejected due to bad request.
static int	CAUSE_CTICCMSIP401UNAUTHORIZED	This cause indicates the request is valid but is not authorized.
static int	CAUSE_CTICCMSIP402PAYMENTREQUIRED	This cause indicates the payment is required for usage.
static int	CAUSE_CTICCMSIP403FORBIDDEN	This cause indicates the server understood the request, but is refusing to fulfill it..
static int	CAUSE_CTICCMSIP404NOTFOUND	This cause indicates the request URI cannot be located by the server.
static int	CAUSE_CTICCMSIP405METHODNOTALLOWED	This cause indicates the method specified in the Request-Line is understood, but not allowed for the address identified by the Request-URI.
static int	CAUSE_CTICCMSIP406NOTACCEPTABLE	This cause indicates the request cannot be proccessed due to requirements in the request cannot be met.
static int	CAUSE_CTICCMSIP407PROXY AUTHENTICATIONREQUIRED	This cause indicates that requset is not authorized and proxy authentication is required for the operation.
static int	CAUSE_CTICCMSIP408REQUESTTIMEOUT	This cause indicates the time out error for the request.
static int	CAUSE_CTICCMSIP410GONE	This cause indicates the requested resource is no longer available at the server and no forwarding address is known.
static int	CAUSE_CTICCMSIP411LENGTHREQUIRED	This cause indicates that an interworking message length is required.
static int	CAUSE_CTICCMSIP413REQUESTENTITY TOOLONG	This cause indicates that the server is refusing to process a request because the request entity-body is larger than the server is willing or able to process.
static int	CAUSE_CTICCMSIP414REQUESTURI TOOLONG	This cause indicates that the server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

Interface	Field	Description
static int	CAUSE_CTICCMSIP415UNSUPPORTED MEDIATYPE	This cause indicates the server is refusing to service the request because the message body of the request indicates the Media Type which is not supported by the server for the requested method.
static int	CAUSE_CTICCMSIP416UNSUPPORTED URISCHEME	This cause indicates the server cannot process the request because the scheme of the URI in the Request-URI is unknown to the server.
static int	CAUSE_CTICCMSIP420BADEXTENSION	This cause indicates the server did not understand the protocol extension specified in a Proxy-Require or Require header field.
static int	CAUSE_CTICCMSIP421EXTENSTION REQUIRED	This cause indicates the UAS needs a particular extension to process the request, but this extension is not listed in a Supported header field in the request.
static int	CAUSE_CTICCMSIP423INTERVALTOOBRIEF	This cause indicates that the server is rejecting the request because the expiration time of the resource refreshed by the request is too short.
static int	CAUSE_CTICCMSIP480TEMPORARILY UNAVAILABLE	This cause indicates the callee's end system was contacted successfully but the callee is currently unavailable (for example, is not logged in, logged in but in a state that precludes communication with the callee, or has activated the "do not disturb" feature).
static int	CAUSE_CTICCMSIP481CALLLEGDOES NOTEXIST	This cause indicates the the UAS received a request that does not match any existing dialog or transaction.
static int	CAUSE_CTICCMSIP482LOOPDETECTED	This cause indicates that the server has detected a loop.
static int	CAUSE_CTICCMSIP483TOOMANYHOOPS	This cause indicates the server received a request that contains a Max-Forwards header field with the value zero (or less than actual hops).
static int	CAUSE_CTICCMSIP484ADDRESS INCOMPLETE	This cause indicates that the server received a request with a Request-URI that was incomplete.
static int	CAUSE_CTICCMSIP485AMBIGUOUS	This cause indicates that the Request-URI was ambiguous.
static int	CAUSE_CTICCMSIP486BUSYHERE	This indicates that the callee's end system was contacted successfully, but the callee is currently not willing or able to take additional calls at this end system.
static int	CAUSE_CTICCMSIP487REQUEST TERMINATED	This cause indicates the request was terminated by a BYE or CANCEL request.

Interface	Field	Description
static int	CAUSE_CTICCMSIP488NOTACCEPTABLE HERE	This cause indicates the same meaning as 606 (Not Acceptable), but only applies to the specific resource addressed by the Request-URI and the request may succeed elsewhere.
static int	CAUSE_CTICCMSIP491REQUESTPENDING	This cause indicates the request was received by a UAS that had a pending request within the same dialog.
static int	CAUSE_CTICCMSIP493UNDECIPHERABLE	This cause indicates that the request was received by a UAS that contained an encrypted MIME body for which the recipient does not possess or will not provide an appropriate decryption key.
static int	CAUSE_CTICCMSIP500SERVERINTERNAL ERROR	This cause indicates the server encountered an unexpected condition that prevented it from fulfilling the request.
static int	CAUSE_CTICCMSIP501NOTIMPLEMENTED	This cause indicates the server does not support the functionality required to fulfill the request.
static int	CAUSE_CTICCMSIP502BADGATEWAY	This cause indicates the server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfill the request.
static int	CAUSE_CTICCMSIP503SERVICEUNAVAILABLE	This cause indicates the server is temporarily unable to process the request due to a temporary overloading or maintenance of the server.
static int	CAUSE_CTICCMSIP504SERVERTIMEOUT	This cause indicates the server did not receive a timely response from an external server it accessed in attempting to process the request.
static int	CAUSE_CTICCMSIP505SIPVERSIONNOT SUPPORTED	This cause indicates the server does not support, or refuses to support, the SIP protocol version that was used in the request.
static int	CAUSE_CTICCMSIP513MESSAGETOOLARGE	This cause indicates the server was unable to process the request since the message length exceeded its capabilities.
static int	CAUSE_CTICCMSIP600BUSYEVERYWHERE	This cause indicates the callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time.
static int	CAUSE_CTICCMSIP603DECLINE	This cause indicates the callee's machine was successfully contacted but the user explicitly does not wish to or cannot participate.
static int	CAUSE_CTICCMSIP604DOESNOTEXIST ANYWHERE	This cause indicates the server has authoritative information that the user indicated in the Request-URI does not exist anywhere.

Interface	Field	Description
static int	CAUSE_CTICCMSIP606NOTACCEPTABLE	This cause indicates the user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable.
static int	CAUSE_CTICONFERENCEFULL	This cause indicates the Conference Call is full and no more participants can be added to it.
static int	CAUSE_CTIDEVICENOTPREEMPTABLE	This cause indicates that the device cannot be preempted.
static int	CAUSE_CTIDROPCONFEREE	This cause indicates the disconnection because the party was dropped from conference.
static int	CAUSE_CTIMANAGER_FAILURE	This cause indicates the failure due to CTI Manager Failure.
static int	CAUSE_CTIPRECEDENCECALLBLOCKED	This cause indicates that there are no predictable circuits or that the called user is busy with a call of equal or higher preventable level.
static int	CAUSE_CTIPRECEDENCELEVELEXCEEDED	This cause indicates that the precedence level of the call has exceeded the authorized level.
static int	CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH	This cause indicates the precedence call has hit low bandwidth and cannot proceed.
static int	CAUSE_CTIPREEMPTFORREUSE	This cause indicates that the call is being preempted and the circuit is reserved for reuse by the preempting exchange.
static int	CAUSE_CTIPREEMPTNOREUSE	This cause indicates the call is being preempted.
static int	CAUSE_DESTINATIONOUTOFORDER	This cause indicates that the destination indicated by the user cannot be reached because the interface to the destination is not functioning correctly.
static int	CAUSE_DESTNUMMISSANDDCNOTSUB	This cause indicates that the specified CUG does not exist.
static int	CAUSE_DPARK	It indicates the call is Directed-Parked call.
static int	CAUSE_DPARK_REMINDER	It indicates the call is Directed Park Reminder call.
static int	CAUSE_DPARK_UNPARK	It indicates that Directed Parked call is now unparked.
static int	CAUSE_EXCHANGEROUTINGERROR	This cause indicates that the exchange couldnt route the call to specified destination.
static int	CAUSE_FAC_CMC	It indicates the FAC(Force Authorization Code) or CMC(Client Matter Code) is needed to route the call.
static int	CAUSE_FACILITYREJECTED	This cause is returned when a supplementary service requested by the user cannot be provided by the network.

Interface	Field	Description
static int	CAUSE_IDENTIFIEDCHANDOESNOTEXIST	This cause indicates that the equipment sending this cause has received a request to use a channel not activated on the interface for a call.
static int	CAUSE_IENIMPL	This cause indicates that the equipment sending this cause has received a message which includes information element(s)/parameter(s) not recognized because the information element(s)/parameter name(s) are not defined or are defined but not implemented by the equipment sending the cause.
static int	CAUSE_INBOUNDBLINDTRANSFER	It indicates the call is IN bound Blind Transfer call.
static int	CAUSE_INBOUNDCONFERENCE	It indicates the call is IN bound Conference call.
static int	CAUSE_INBOUNDTRANSFER	It indicates the call is IN bound Transfer call.
static int	CAUSE_INCOMINGCALLBARRED	This cause indicates that the incoming calls for that number is barred.
static int	CAUSE_INCOMPATABLEDDESTINATION	This cause indicates that the equipment sending this cause has received a request to establish a call which has low layer compatibility.
static int	CAUSE_INTERWORKINGUNSPECIFIED	This cause indicates that an interworking call has ended.
static int	CAUSE_INVALIDCALLREFVALUE	This cause indicates that the equipment sending this cause has received a message with a call reference which is not currently in use on the user-network interface.
static int	CAUSE_INVALIDIECONTENTS	This cause indicates that the equipment sending this cause has received and information element which it has implemented; however, one or more of the fields in the information element are coded in such a way which has not been implemented by the equipment sending this cause.
static int	CAUSE_INVALIDMESSAGEUNSPECIFIED	This cause is used to report an invalid message event only when no other cause in the invalid message class applies.
static int	CAUSE_INVALIDNUMBERFORMAT	This cause indicates that the called party cannot be reached because the called party number is not in a valid format or is not complete.
static int	CAUSE_INVALIDTRANSITNETSEL	This cause indicates that a transit network identification was received which is of an incorrect format.
static int	CAUSE_MANDATORYIEMISSING	This cause indicates that the equipment sending this cause has received a message which is missing an information element which must be present in the message before that message can be processed.

Interface	Field	Description
static int	CAUSE_MSGNCOMPATABLEWCS	This cause indicates that a message has been received which is incompatible with the call state.
static int	CAUSE_MSGTYPENCOMPATWCS	This cause indicates that the equipment sending this cause has received a message such that the procedures do not indicate that this is a permissible message to receive while in the call state, or a STATUS message was received indicating an incompatible call state.
static int	CAUSE_MSGTYPENIMPL	This cause indicates that the equipment sending this cause has received a message with a message type it does not recognize either because this is a message not defined or defined but not implemented by the equipment sending this cause.
static int	CAUSE_NETOUTOFORDER	This cause indicates that the network is not functioning correctly and that the condition is likely to last a relatively long period of time.
static int	CAUSE_NOANSWERFROMUSER	This cause is used when the called party has been alerted but does not respond with a connect indication within a prescribed period of time.
static int	CAUSE_NOCALLSUSPENDED	This cause indicates that the network has received a call resume request containing a call identity information element which presently does not indicate any suspended call within the domain of interfaces over which calls may be resumed.
static int	CAUSE_NOCIRCAVAIL	This cause indicates that there is no appropriate circuit/channel presently available to handle the call.
static int	CAUSE_NOERROR	This is usually given when there is no error and operation completes successfuly.
static int	CAUSE_NONSELECTEDUSERCLEARING	This cause indicates that the user has not been awarded the incoming call.
static int	CAUSE_NORMALCALLCLEARING	This cause indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared.
static int	CAUSE_NORMALUNSPECIFIED	This cause is used to report a normal event only when no other cause in the normal class applies.
static int	CAUSE_NOROUTETODDESTINATION	This cause indicates that the called party cannot be reached because the network through which the call has been routed does not serve the destination desired.
static int	CAUSE_NOROUTETOTRANSITNET	This cause indicates that the equipment sending this cause has received a request to route the call through a particular transit network which it does not recognize.

Interface	Field	Description
static int	CAUSE_NOUSERRESPONDING	This cause is used when a called party does not respond to a call establishment message with either an alerting or connect indication within the prescribed period of time allocated.
static int	CAUSE_NUMBERCHANGED	This cause is returned to a calling party when the called party number indicated by the calling party is no longer assigned.
static int	CAUSE_ONLYRDIVEARERCAPAVAIL	This cause indicates that the calling party has requested an unrestricted bearer service but the equipment sending this cause only supports the restricted version of the requested bearer capability.
static int	CAUSE_OUTBOUNDCONFERENCE	It indicates the call is OUT bound Conference call.
static int	CAUSE_OUTBOUNDTRANSFER	It indicates the call is OUT bound Transfer call
static int	CAUSE_OUTOFBANDWIDTH	This cause indicates that the call could not proceed because of Low Bandwidth.
static int	CAUSE_PROTOCOLERRORUNSPECIFIED	This cause is used to report a protocol error event only when no other cause in the protocol error class applies.
static int	CAUSE_QSIG_PR	It indicates the QSIG Path Replacement in the call.
static int	CAUSE_QUALOFSERVNAVAIL	This cause is used to report that the requested Quality of Service, as defined in Recommendation X.213.
static int	CAUSE_QUIET_CLEAR	It indicates the Call is cleared as Call Manager has gone down, but media between endpoints remain connected.
static int	CAUSE_RECOVERYONTIMEREXPIRY	This cause indicates that a procedure has been initiated by the expiration of a timer in association with error handling procedures.
static int	CAUSE_REDIRECTED	This cause indicates the call is being redirected to different party.
static int	CAUSE_REQCALLIDHASBEENCLEARED	This cause indicates that the network has received a call resume request containing a call identity information element indicating a suspended call that has in the meantime been cleared while suspended (either by network time-out or by the remote user).
static int	CAUSE_REQCIRCNAVIL	This cause is returned when the circuit or channel indicated by the requesting entity cannot be provided by the other side of the interface.
static int	CAUSE_REQFACILITYNIMPL	This cause indicates that the equipment sending this cause does not support the requested.

Interface	Field	Description
static int	CAUSE_REQFACILITYNOTSUBSCRIBED	This cause indicates that the user has requested a supplementary service which is implemented by the equipment which generated this cause but the user is not authorized to use.
static int	CAUSE_RESOURCESNAVAIL	This cause is used to report a resource unavailable event.
static int	CAUSE_RESPONSETOSTATUSENQUIRY	This cause is included in the STATUS message when the reason for generating the STATUS message was the prior receipt of a STATUS INQUIRY.
static int	CAUSE_SERVNOTAVAILUNSPECIFIED	This cause is used to report a service or option not available event only when no other cause in the service or option not available class applies.
static int	CAUSE_SERVOPERATIONVIOLATED	This cause indicates that although the calling party is a member of the CUG for the outgoing CUG call.
static int	CAUSE_SERVOROPTNAVAILORIMPL	This cause is used to report a service or option not implemented event only when no other cause in the service or option not implemented class applies.
static int	CAUSE_SUBSCRIBERABSENT	This cause value is used when a mobile station has logged off.
static int	CAUSE_SUSPCALLBUTNOTTHISONE	This cause indicates that a call resume has been attempted with a call identity which differs from that in use for any presently suspended call(s).
static int	CAUSE_SWITCHINGEQUIPMENTCONGESTION	This cause indicates that the switching equipment generating this cause is experiencing a period of high traffic.
static int	CAUSE_TEMPORARYFAILURE	This cause indicates that the network is not functioning correctly and that the condition is not likely to last a long period of time; e.g., the user may wish to try another call attempt almost immediately.
static int	CAUSE_UNALLOCATEDNUMBER	This cause indicates that the destination requested by the calling user cannot be reached because, it is an invalid number.
static int	CAUSE_USERBUSY	This cause is used to indicate that the called party is unable to accept another call because the user busy condition has been encountered.

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Methods

Table 81: Methods in CiscoCallEv

Interface	Method	Description
Int	getCiscoCause()	Returns the Cisco Unified Communications Manager cause for this event. To function properly, some applications need to know the reason why an event happened at an endpoint that the application is observing. For example, a Connection may be disconnected because the call was not answered (CAUSE_NOANSWERFROMUSER), or whether the caller it was disconnected because it was rejected (CAUSE_CALLREJECTED). Returns: The Cisco Unified Communications Manager cause for this event

Interface	Method	Description
Int	getCiscoFeatureReason()	Returns the Cisco Unified Communications Manager Feature Reason for this event. To function properly, some applications need to know the reason why an event happened. This interface provides the CiscoFeatureReason in JTAPI Call events for current and new features. Existing features, such as transfer, continue to receive the CiscoCause provided by the older interface CiscoCallEv.getCiscoCause(), while this interface will provide REASON_TRANSFER for transfer. Caution: Applications should make sure to handle unrecognized reasons and provide default behavior, because new reasons could be added in the future and this interface may not be backward compatible. The possible values are defined in the CiscoFeatureReason interface. Returns: The Cisco Unified Communications Manager Feature Reason for this event

Related Documentation

See Constant Field Values, on page 1661 and CallEv for more information.

CiscoCallFeatureCancelledEv

This event notifies applications that the cancel operation has been invoked

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(2)	Created history table to track changes.

Declaration

public interface CiscoCallFeatureCancelledEv

Methods

Table 82: Methods in CiscoCallFeatureCancelledEv

Interface	Method	Description
CiscoCall	getConsultCall()	Returns the Consult Call for which consult operation is cancelled, if the consult call doesn't exist it will return NULL.

Related Documentation

See Constant Field Values, on page 1661.

CiscoCallID

The CiscoCallID object represents a unique object that is associated with each call. Applications may use the object itself or the integer representation of the object that the intValue() method returns.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

CiscoObjectContainer

Declaration

Public interface CiscoCallID extends CiscoObjectContainer

Fields

None

Methods

Table 83: Methods in CiscoCallID

Interface	Method	Description
Int	intValue()	Returns an integer representation of this object. Returns: Int An integer representation of this object
CiscoCall	getCall()	Returns the CiscoCall corresponding to this CiscoCallID.
int	getCallManagerID()	Returns the Cisco Unified Communications Manager NodeID of the call associated with this CiscoCallID.
int	getGlobalCallID()	Returns the GlobalCallID of the call associated with this CiscoCallID.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

None

CiscoMediaCallSecurityIndicator

CiscoMediaCallSecurityIndicator lets you retrieve the security indicator for a call.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoMediaCallSecurityIndicator

Fields

None

Methods

Table 84: Methods in CiscoMediaCallSecurityIndicator

Interface	Method	Description
CiscoCallID	getCallID()	Returns the CiscoCallID.
int	getCiscoMediaSecurityIndicator()	Returns the media security indicator, one of the following constants: CiscoMediaSecurityIndicator. MEDIA_ENCRYPT_USER_NOT_AUTHORIZED CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_UNAVAILABLE CiscoMediaSecurityIndicator. MEDIA_NOT_ENCRYPTED

Interface	Method	Description
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object.Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event was delivered, CiscoProvider.getCall may return null.

Related Documentation

See CiscoRTPParams.

CiscoCallSecurityStatusChangedEv

Applications receive CiscoCallSecurityStatusChangedEv when the overall Call security status changes.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoCallSecurityStatusChangedEv extends CiscoCallEv

Fields

Table 85: Fields in CiscoCallSecurityStatusChangedEv

Interface	Field
Static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED,

CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION,
CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE,
CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED,
CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE,
CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH,
CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE,
CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK,
CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR,
CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST,
CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL,
CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED,
CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING,
CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED,
CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET,
CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED,
CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,
CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH,
CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL,
CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL,
CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL,
CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED,
CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL,

CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 86: Methods in CiscoCallSecurityStatusChangedEv

Interface	Method	Description
javax.telephony.events.Ev	getID()	Specified by: getID in interface javax.telephony.events.Ev
getCallSecurityStatus	getCallSecurityStatus()	Returns the call security status. This interface can return: CiscoCall.CALLSECURITY_UNKNOWN, CiscoCall.CALLSECURITY_NOTAUTHENTICATED, CiscoCall.CALLSECURITY_AUTHENTICATED, CiscoCall.CALLSECURITY_ENCRYPTED

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See also Constant Field Values, on page 1661 for more information.

CiscoConferenceChain

This interface provides links to conference chain connections for the conference calls that are linked together in a conference chain. You can obtain this object from CiscoConferenceChainAddedEv and CiscoConferenceChainRemovedEv.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Declaration

public interface CiscoConferenceChain

Fields

None

Methods

Table 87: Methods in CiscoConferenceChain

Interface	Method	Description
javax.telephony.Connection[]	getChainedConferenceConnections()	Returns an array of Connections for Conference Calls that are chained together in a single conference. Applications can use this list to get all the Conference Calls that are linked together. To get the list of Connections for all the Calls that are chained together in the Conference, the provider must have an observer on at least one party in every conference call.

Interface	Method	Description
CiscoCall[]	getChainedConferenceCalls()	Returns an array of Calls that are chained together in a single conference. This interface returns only the Calls in the conference chain that are observed in the provider.

Related Documentation

See CiscoConferenceChainAddedEv and CiscoConferenceChainRemovedEv for more information.

CiscoConferenceChainAddedEv

The system sends a CiscoConferenceChainAddedEv event when a conference chain connection gets added to a call. This event gets sent every time a new conference chain connection gets added. This event gets reported via theCallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

All Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoConferenceChainAddedEv extends CiscoCallEv

Fields

Table 88: Fields in CiscoConferenceChainAddedEv

Interface	Field
static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 89: Methods in CiscoConferenceChainAddedEv

Interface	Method	Description
javax.telephony.Connection	getAddedConnection()	Returns the conference chain Connection that was added to the call.
CiscoConferenceChain	getConferenceChain()	Returns a CiscoConferenceChain that contains all of the conference connections for the calls that are chained together.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoConferenceChainRemovedEv

The system sends a CiscoConferenceChainRemovedEv event when a conference chain connection gets removed from a call. This event gets sent whenever a conference chain connection gets removed. This event gets reported via theCallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoConferenceChainRemovedEv extends CiscoCallEv

Fields

Table 90: Fields in CiscoConferenceChainRemovedEv

Interface	Field
static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 91: Methods in CiscoConferenceChainRemovedEf

Interface	Method	Description
CiscoConferenceChain	getConferenceChain()	Returns a CiscoConferenceChain that contains all of the conference connections for the calls that are chained together. Returns: Connection.
javax.telephony.Connection	getRemovedConnection()	Returns the conference chain Connection that was removed from the call. Returns: CiscoConferenceChain.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoConferenceEndEv

The CiscoConferenceEndEv event indicates that a Conference operation completed. The system reports this event via the CallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoConferenceEndEv extends CiscoCallEv

Fields

Table 92: Fields in CiscoConferenceEndEv

Interface	Field
static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 93: Methods in CiscoConferenceEndEv

Interface	Method	Description
javax.telephony.Address	getConferenceControllerAddress()	Returns the Address that currently acts as the conference controller for this call, the initiating call.
javax.telephony.Call	getConferencedCall()	Returns the call that merged. This call is in the Call.INVALID state.
javax.telephony.Call[]	getFailedCalls()	Returns list of Calls that could not be Conferenced.
javax.telephony.Call	getFinalCall()	Returns the call that remains active after the conference completes.

Interface	Method	Description
javax.telephony.TerminalConnection	getHeldConferenceController()	Returns the TerminalConnection that currently acts as the conference controller for this call -- the final call. This is the TerminalConnection that was in HELD state when the conference got initiated. This method returns null or TerminalConnection if the conference controller is not being observed.
javax.telephony.TerminalConnection	getTalkingConferenceController()	Returns the TerminalConnection that currently acts as the conference controller for this call -- the initiating call.This is the TerminalConnection that was in TALKING state. This method returns null or TerminalConnection if the conference controller is not being observed.
boolean	isSuccess()	Returns Boolean True or False depending on whether the conference succeeded or failed. The application can use this interface to determine whether a Conference is successful. Conferences will fail in these situations: • If the application issues the request Call.conference(otherCalls[]), the system considers the conference as failed if one or more than one Calls could not Join into Conference. Use getFailedCalls() to find the failed calls. • If no conference bridge is available, and the conference could not complete. Use getFailedCalls() to get a list of the calls that could not join the conference. • If the party being conferenced drops out before the conference could complete.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661

See also isSuccess()

CiscoConferenceStartEv

The CiscoConferenceStartEv event indicates that a conference operation started. The CallControlCallObserver interface reports this event.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added getControllerTerminalName() method for Join Across Lines/Connected Conference feature.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoConferenceStartEv extends CiscoCallEv

Fields

Table 94: Fields in CiscoConferenceStartEv

Interface	Field
static int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 95: Methods in CiscoConferenceStartEv

Interface	Method	Description
javax.telephony.Address	getConferenceControllerAddress()	Returns the Address that currently acts as the conference controller for this call, the initiating call.
javax.telephony.Call	getConferencedCall()	Returns the call that will be conferenced. This is the call that will be merged into the initiating call. This interface returns the first call from the list of calls that are joining into conference.
javax.telephony.Call[]	getConferencedCalls()	Returns the list of the calls that will be conferenced. These calls are the ones that will be merged into the final call.

Interface	Method	Description
javax.telephony.Call	getFinalCall()	Returns the call that will remain active after the conference completes. This is the call into which all the calls will merge.
javax.telephony.TerminalConnection	getHeldConferenceController()	Returns the TerminalConnection that currently acts as the conference controller for this call, the initiating call. This is the TerminalConnection that was in HELD state. This method returns null if the conference controller is not being observed. This method returns the first held controller for a multiple call join scenario.
javax.telephony.TerminalConnection[]	getHeldConferenceControllers()	Returns all TerminalConnections on Conference Controller Terminal that are joining together and are in HELD State.
javax.telephony.Address	getOriginalConferenceControllerAddress()	Returns the Address of the participant that initiated the conference.
javax.telephony.TerminalConnection	getTalkingConferenceController()	Returns the TerminalConnection that currently acts as the conference controller for this call, the initiating call. This is the TerminalConnection that was in TALKING state. This method returns null if the conference controller is not being observed. This method returns null if there is no controller in talking state. Calls can be joined into a conference without any talking controller.
String	getControllerTerminalName()	Returns the terminal name of the controllers across which a conference is done.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoConnection

The CiscoConnection interface extends the CallControlConnection interface with additional Cisco specific capabilities. Applications can use the getReason method to obtain the reason for the creation of a connection.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added two new methods: getPartyInfo and disconnect(CiscoPartyInfopartyInfo for Drop Any Party feature.
8.0(1)	Enhanced with the following: • New method to get the associated CiscoHuntConnection. If application is observing hunt list member, applications can use this method to find out if call is routed through HuntPilot. • New interface getUniqueID(Terminal term) is added which will return the uniqueID as string. • New method that allows an application to determine if the connection is associated with a chaperone device on a chaperone call. Chapone devices have a limited feature set, and knowing that a connection is associated with a chaperone device can allow the application to better handle the connections.
11.5(1)	Added redirect method with deviceName.

All Superinterfaces

javax.telephony.callcontrol.CallControlConnection, CiscoObjectContainer, javax.telephony.Connection

Declaration

public interface CiscoConnection extends javax.telephony.callcontrol.CallControlConnection, CiscoObjectContainer

Fields

Table 96: Fields in CiscoConnection

Interface	Field	Description
static int	ADDRESS_SEARCH_SPACE	The redirect should be done by using the redirect controller address search space.
static int	CALLED_ADDRESS_DEFAULT	The default behavior for Cisco JTAPI should apply.
static int	CALLED_ADDRESS_SET_TO_PREFERRED CALLEDPARTY	The original called Address should be set to the value present in preferredOriginalCalledParty field.
static int	CALLED_ADDRESS_SET_TO_REDIRECT_ DESTINATION	The called Address should be reset to the redirect destination.
static int	CALLED_ADDRESS_UNCHANGED	The called Address should remain unchanged after the redirect operation.
static int	CALLINGADDRESS_SEARCH_SPACE	The redirect should be done by using the calling address search space.
static int	DEFAULT_SEARCH_SPACE	The redirect should be done by using the default search space for the implementation.
static int	REASON_DIRECTCALL	This Connection results from a direct call.
static int	REASON_FORWARDALL	This Connection results from unconditional forwarding.
static int	REASON_FORWARDBUSY	This Connection results from a forwarding on busy.
static int	REASON_FORWARDNOANSWER	This Connection results from a forwarding on no answer.
static int	REASON_OUTBOUND	This Connection is an originating Connection, not a destination Connection.
static int	REASON_REDIRECT	This Connection results from a redirection.

Interface	Field	Description
static int	REASON_TRANSFERREDCALL	This Connection results from a transfer.
static int	REDIRECT_DROP_ON_FAILURE	This redirect mode instructs the implementation to perform redirect without checking the validity or availability of the destination.
static int	REDIRECT_NORMAL	This redirect mode instructs the implementation to perform redirect if the destination is valid and available.

Inherited Fields

From Interface javax.telephony.callcontrol.CallControlConnection

ALERTING, DIALING, DISCONNECTED, ESTABLISHED, FAILED, IDLE, INITIATED, NETWORK_ALERTING, NETWORK_REACHED, OFFERED, OFFERING, QUEUED, UNKNOWN

From Interface javax.telephony.Connection

CONNECTED, INPROGRESS

Methods

Table 97: Methods in CiscoConnection

Interface	Method and Descrption
Boolean	getAddressPI()
	Returns Presentation Indicator (PI) associated with the Address on which the connection is created.
CiscoHuntConnection	getCiscoHuntConnection()
	This method returns the associated CiscoHuntConnection or null.
CiscoConnectionID	getConnectionID()
	Returns CiscoConnectionID for this CiscoConnection
java.lang.String	getDParkPrefixCode()
	Returns the prefix code that needs to be dialed with the DPark DN to retrieve the call.

Interface	Method and Descrption
int	getReason()
	Returns the reason for the creation of this Connection. To function properly, some applications need to know the reason for the creation of the connection is created at an endpoint. The reason for a Connection creation may be any of the following constants: • CiscoConnection. REASON_DIRECTCALL CiscoConnection. REASON_TRANSFERREDCALL • CiscoConnection. REASON_FORWARDNOANSWER • CiscoConnection. REASON_FORWARDBUSY • CiscoConnection. REASON_FORWARDALL • CiscoConnection. REASON_REDIRECT • CiscoConnection. REASON_NORMAL
javax.telephony.TerminalConnection	getRequestController()
	Returns the current request Controller for the Connection.
String	getUniqueID(Terminal term)
	This method returns the updated uniqueID of the connection. In case if there are no shared lines associated with this connection, application can just pass null object as parameter to this interface to get the Unique Identifier. Unique Identifier will be same for all the shared lines, but if it's a barge scenario, different terminals would have different identifier.The returned Unique Identifier will be 32-character hex string. Please refer to End to End Call Tracing, on page 898End to End Call Tracing, on page 898, for more information. **Throws:** PrivilegeVoilationException , InvalidStateException **Parameters:** Terminal
boolean	isChaperone()
	This method returns true if the connection is associated with a Chaperone call, and false if not.
java.lang.String	park()
	This method parks the call at a system park DN and returns the address of the park DN.

Interface	Method and Descrption
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode)
	This method overloads the CallControlConnection.redirect() method. **Throws** javax.telephony. InvalidStateException javax.telephony. InvalidPartyException javax.telephony. MethodNotSupportedException javax.telephony. PrivilegeViolationException javax.telephony. ResourceUnavailableException **Parameter** Mode - This parameter can take one of the following two values: • CiscoConnection. REDIRECT_DROP_ON_FAILURE: This mode instructs the implementation to perform a redirect without checking the validity or availability of the destination. The original call gets dropped if the destination is invalid or busy. • CiscoConnection. REDIRECT_NORMAL: This mode instructs the implementation to perform a redirect only after checking the validity or availability of the destination. This matches the behavior of the CallControlConnection.redirect() method. The system does not drop the original call on failure.

Interface	Method and Descrption
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace)
	This method overloads the CallControlConnection.redirect() method. It takes two new parameters: redirectMode and callingSearchSpace. The redirectMode selects which type of redirect to perform. The callingSearchSpace tells the implementation to use either the calling party search space or the redirect controller search space. **Parameters** mode - One of the following: • CiscoConnection.REDIRECT_DROP_ON_FAILURE: This mode instructs the implementation to perform a redirect without checking the validity or availability of the destination. The original call gets dropped if the destination is invalid or busy. • CiscoConnection.REDIRECT_NORMAL: This mode instructs the implementation to perform a redirect only after checking the validity or availability of the destination. This matches the behavior of the CallControlConnection.redirect() method. The system does not drop the original call on failure. callingSearchSpace - One of the following: • CiscoConnection. DEFAULT_SEARCH_SPACE • CiscoConnection. CALLINGADDRESS_SEARCH_SPACE • CiscoConnection. ADDRESS_SEARCH_SPACE **Throws** javax.telephony. InvalidStateException javax.telephony. InvalidPartyException javax.telephony. MethodNotSupportedException javax.telephony. PrivilegeViolationException javax.telephony. ResourceUnavailableException

Interface	Method and Descrption
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption)
	This method overloads the CallControlConnection.redirect() method. It takes three new parameters: mode, callingSearchSpace, and calledAddressOption. The redirectMode selects the type of redirect to perform. The callingSearchSpace tells the implementation to use either the calling party search space or the redirect controller search space. The calledAddressOption parameter controls whether to reset the original called fields. **Parameters** mode - One of the following: • CiscoConnection. REDIRECT_DROP_ON_FAILURE • CiscoConnection. REDIRECT_NORMAL callingSearchSpace - One of the following: • CiscoConnection. DEFAULT_SEARCH_SPACE • CiscoConnection. CALLINGADDRESS_SEARCH_SPACE • CiscoConnection. ADDRESS_SEARCH_SPACE calledAddressOption: One of the following: • CiscoConnection. CALLED_ADDRESS_DEFAULT • CiscoConnection. CALLED_ADDRESS_UNCHANGED • CiscoConnection. CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION **Throws** javax.telephony. InvalidStateException javax.telephony. InvalidPartyException javax.telephony. MethodNotSupportedException javax.telephony. PrivilegeViolationException javax.telephony. ResourceUnavailableException

Interface	Method and Descrption
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, java.lang.String preferredOriginalCalledParty, java.lang.String facCode, java.lang.String cmcCode)

Interface	Method and Descrption
	This method overloads the CallControlConnection.redirect() method. It takes three new parameters: mode, callingSearchSpace, and calledAddressOption. The redirectMode selects the type of redirect to perform. The callingSearchSpace tells the implementation to use either the calling party search space or the redirect controller search space. The calledAddressOption parameter controls whether to reset the original called fields. If the FAC and CMC codes are missing or invalid, the call might not get offered and platformException may contain one of the following error codes: • CiscoJTAPIException. CTIERR_FAC_CMC _REASON_FAC_NEEDED • CiscoJTAPIException. CTIERR_FAC_CMC _REASON_CMC_NEEDED • CiscoJTAPIException. CTIERR_FAC_CMC _REASON_FAC_CMC_NEEDED • CiscoJTAPIException. CTIERR_FAC_CMC _REASON_FAC_INVALID • CiscoJTAPIException. CTIERR_FAC_CMC _REASON_CMC_INVALID **Parameters** mode - One of the following: • CiscoConnection.REDIRECT_DROP_ON_FAILURE • CiscoConnection. REDIRECT_NORMAL callingSearchSpace - One of the following: • CiscoConnection. DEFAULT_SEARCH_SPACE • CiscoConnection. CALLINGADDRESS_SEARCH_SPACE • CiscoConnection. ADDRESS_SEARCH_SPACE • preferredOriginalCalledParty - may be a DN that will be the originalCalledParty field when call is offered to destinationAddress. If this field * needs to be used, applications must set calledAddressOption as CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY. If applications are not interested in this field, you must pass the default value of null. calledAddressOption - One of the following: • CiscoConnection. CALLED_ADDRESS_DEFAULT • CiscoConnection. CALLED_ADDRESS_UNCHANGED • CiscoConnection. CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION preferredOriginalCalledParty - may be a DN that will be the originalCalledParty field when call is offered to destinationAddress. If this field * needs to be used, applications must set calledAddressOption as CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY. If applications are not interested in this field, you must pass the default value of null. facCode - required if the destinationAddress requires a forced authorization

Interface	Method and Descrption
	code to offer the call. Pass the FAC in this parameter. Pass the default value of null if the destinationAddress does not require a FAC code. cmcCode - required if the destinationAddress requires a client matter code to offer the call. Pass the CMC in this parameter. Pass the default value of null if the destinationAddress does not require a CMC code.
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, java.lang.String preferredOriginalCalledParty, java.lang.String facCode, java.lang.String cmcCode, int featurePriority)
	This method overloads CallControlConnection.redirect(). It takes a new parameter, featurePriority, that sets the call priority. The featurePriority parameter may be: • CiscoCall.FEATUREPRIORITY_NORMAL • CiscoCall.FEATUREPRIORITY_URGENT • CiscoCall.FEATUREPRIORITY_EMERGENCY **Returns** Connection **Throws** javax.telephony.InvalidStateException javax.telephony.InvalidPartyException javax.telephony.MethodNotSupportedException javax.telephony.PrivilegeViolationException javax.telephony.ResourceUnavailableException

Interface	Method and Descrption
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, java.lang.String preferredOriginalCalledParty)
	This method overloads the CallControlConnection.redirect() method. It takes three new parameters: mode, callingSearchSpace, and preferredOriginalCalledParty. The redirectMode selects the type of redirect to perform. The callingSearchSpace tells the implementation to use either the calling party search space or the redirect controller search space. **Parameters** mode - One of the following: • CiscoConnection.REDIRECT_DROP_ON_FAILURE • CiscoConnection.REDIRECT_NORMAL callingSearchSpace - One of the following: • CiscoConnection.DEFAULT_SEARCH_SPACE • CiscoConnection.CALLINGADDRESS_SEARCH_SPACE • CiscoConnection.ADDRESS_SEARCH_SPACE preferredOriginalCalledParty - May be a DN that will be the originalCalledParty field when the call gets offered to the destinationAddress. **Throws** javax.telephony.InvalidStateException javax.telephony.InvalidPartyException javax.telephony.MethodNotSupportedException javax.telephony.PrivilegeViolationException javax.telephony.ResourceUnavailableException

Interface	Method and Descrption
javax.telephony.Connection	redirect(String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, String preferredOriginalCalledParty, String facCode, String cmcCode, int featurePriority, byte[] applicationXMLData)
	This method is similar to the existing redirect method on the CiscoConnection object, except this one takes an additional parameter, applicationXMLData. Parameters applicationXMLData This parameter was added, and it allows an application to send message header point like SIP contact header info to the receiving end point. The parameter takes xml format as mentioned below. <data> <item> <type>contact</type> <operation>append</operation> <protocol>SIP</protocol> <value>;+sip.instance = "<urn:uuid = *guid*>"</value> </item> </data> **Note** This version only supports: contact, operation: append, protocol: SIP. It can be enhanced to support other protocols and operations in the future. If applications are not interested in this field, you must pass the default value of null.
javax.telephony.Connection	redirect(String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, String preferredOriginalCalledParty, String facCode, String cmcCode, int featurePriority, byte[] applicationXMLData, String deviceName)
	This method is similar to the above method, but it adds the deviceName parameter, which allows you to send the redirect to a specific device. Even in situations where the target device shares a line with another device, the redirected call goes only to the target device and not to the other device that shares the phone line.
void	setRequestController(javax.telephony.TerminalConnection tc)
	This interface gets provided to a requesting TerminalConnection.
com.cisco.jtapi.extensions.CiscoPartyInfo[]	getPartyInfo()
	Returns a list of participants.

Interface	Method and Descrption
java.lang.void	disconnect(CiscoPartyInfopartyInfo)
	Disconnects participant with whose CiscoPartyInfo matches the passed parameter value; throws exception otherwise. **Throws** PrivilegeViolationException InvalidStateException
getLocalUUID(TerminalConnection termConn)	This method takes a Terminal connection object of the connection and returns the Local Universal Unique Identifier of the party associated with both connection and the terminal connection.
getPeerUUID(TerminalConnection termConn)	This method takes a Terminal connection object of the connection and returns the Local Universal Unique Identifier of the party on the other side of the call. It is a part of both connection and the terminal connection.

Inherited Methods

From Interface javax.telephony.callcontrol.CallControlConnection

accept, addToAddress, getCallControlState, park, redirect, reject

From Interface javax.telephony.Connection

disconnect, getAddress, getCall, getCapabilities, getConnectionCapabilities, getState, getTerminalConnections

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Documentation

None

CiscoConnectionID

The CiscoConnectionID object represents a unique object that is associated with each connection. Applications may use the object itself or the integer representation of the object that the intValue() method returns.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoObjectContainer

Declaration

public interface CiscoConnectionID extends CiscoObjectContainer

Fields

None

Methods

Table 98: Methods in CiscoConnectionID

Interface	Method	Description
CiscoConnection	getConnection()	Returns the CiscoConnection for the CiscoConnectionID.
Int	intValue()	Returns an integer representation of this object.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

None

CiscoConnectionUniqueIDChangedEv

It's a new event to highlight that uniqueID of the connection has changed.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New event

Declaration

public interface CiscoConnectionUniqueIDChangedEv extends ConnEv

Methods

Table 99: Methods in CiscoConnectionUniqueIDChangedEv

Interface	Method	Description
String	getOldUniqueID()	This method returns the old uniqueID of the connection which has just changed. The returned value is a Unique Identifier as 32-character hex string.
Terminal	getTerminal()	This method returns the Terminal for which this ConnEv is delivered.
String	getUniqueID()	This method returns the updated uniqueID of the connection. The returned value is a Unique Identifier as 32-character hex string.

Related Documentation

CiscoConsultCall

The CiscoConsultCall interface extends the CiscoCall interface to expose certain properties of calls that have been created as part of a consultative transfer or consultative conference.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.Call, javax.telephony.callcontrol.CallControlCall, CiscoCall, CiscoObjectContainer

Declaration

public interface CiscoConsultCall extends CiscoCall

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCall

CALLSECURITY_AUTHENTICATED, CALLSECURITY_ENCRYPTED, CALLSECURITY_NOTAUTHENTICATED, CALLSECURITY_UNKNOWN, FEATUREPRIORITY_EMERGENCY, FEATUREPRIORITY_NORMAL, FEATUREPRIORITY_URGENT, PLAYTONE_BOTHLOCALANDREMOTE, PLAYTONE_LOCALONLY, PLAYTONE_NOLOCAL_OR_REMOTE, PLAYTONE_REMOTEONLY, SILENT_MONITOR

From Interface javax.telephony.Call

ACTIVE, IDLE, INVALID

Methods

Table 100: Methods in CiscoConsultCall

Interface	Method	Description
javax.telephony.TerminalConnection	getConsultingTerminalConnection()	Returns the consulting TerminalConnection that was used to create this CiscoConsultCall. If this Call was created as part of a consultative transfer or consultative conference, the getConsultingTerminalConnection method returns the TerminalConnection that was used to perform the consultation on the original call. This method lets you correlate a ConsultCall with its original call. The original call itself does not have any methods that you can use determine the ConsultCall, if any, to which it is related. Returns: Null if this Call does not result from a consultation, or the consulting TerminalConnection of the original Call if this call resulted from a consultation.

Interface	Method	Description
javax.telephony.Connection[]	consultWithoutMedia(javax.telephony.TerminalConnection tc, java.lang.String dialedDigits)	Provides applications ability to initiate a consultative call without setting up media for it. This interface may be invoked when application is creating a consult call and completing transfer before media establishes for consult call. Cisco Unified Communication Manager may some times run into erroneous race condition when consult call is answered, and application completes transfer in the middle of media setup for consult call. To avoid this problem, application that does not wait for media setup completion for consult call, may use this method to setup consult call. From CallEvent perspective, this method behaves similar to CallControlCall.consult(TerminalConnection tc, String dialedDigits). Creates a consultation between this Call and an active Call without establishing the media. This consult call may only be transferred, not conferenced. Cisco JTAPI does not support this method with CallControlCall.setConferenceEnable(). Cisco JTAPI only supports this method with CallControlCall.setTransferEnable(). **Throws** javax.telephony. InvalidStateException javax.telephony. InvalidArgumentException javax.telephony. MethodNotSupportedException javax.telephony. ResourceUnavailableException javax.telephony. PrivilegeViolationException javax.telephony. InvalidPartyException

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCall

conference, connect, getCalledAddressPI, getCalledPartyInfo, getCallID, getCallingAddressPI, getCallSecurityStatus, getConferenceChain, getCurrentCalledAddress, getCurrentCalledAddressPI, getCurrentCalledDisplayNamePI, getCurrentCalledPartyDisplayName, getCurrentCalledPartyInfo, getCurrentCalledPartyUnicodeDisplayName, getCurrentCalledPartyUnicodeDisplayNamelocale, getCurrentCallingAddress, getCurrentCallingAddressPI, getCurrentCallingDisplayNamePI, getCurrentCallingPartyDisplayName, getCurrentCallingPartyInfo, getCurrentCallingPartyUnicodeDisplayName, getCurrentCallingPartyUnicodeDisplayNamelocale, getGlobalizedCallingParty, getLastRedirectedPartyInfo, getLastRedirectingAddressPI, getLastRedirectingPartyInfo, getModifiedCalledAddress, getModifiedCallingAddress, startMonitor, startMonitor, transfer

From Interface javax.telephony.callcontrol.CallControlCall

addParty, conference, consult, consult, drop, getCalledAddress, getCallingAddress, getCallingTerminal, getConferenceController, getConferenceEnable, getLastRedirectedAddress, getTransferController, getTransferEnable, offHook, setConferenceController, setConferenceEnable, setTransferController, setTransferEnable, transfer, transfer

From Interface javax.telephony.Call

addObserver, connect, getCallCapabilities, getCapabilities, getConnections, getObservers, getProvider, getState, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

See CiscoCall for more information.

CiscoConsultCallActiveEv

The CiscoConsultCallActiveEv event interface extends the JTAPI CallActiveEv event. This event indicates that the state of the call object changed to Call.ACTIVE and that the call was initiated as a result of a consultative transfer or consultative conference operation (manual or programmatic). Applications can obtain the consulting TerminalConnection on the original (consulting) call by using the CiscoConsultCall.getConsultingTerminalConnection method.

The system reports this event to applications via the CallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallActiveEv, javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoConsultCallActiveEv extends CiscoCallEv, javax.telephony.events.CallActiveEv

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

javax.telephony.TerminalConnectiongetHeldTerminalConnection() Deprecated. Replaced by CiscoConsultCall.getConsultingTerminalConnection()

Table 101: Methods in CiscoConsultCallActiveEv

Interface	Method	Description
javax.telephony.TerminalConnection	getHeldTerminalConnection()	Deprecated method. Replaced by CiscoConsultCall. GetConsultingTerminalConnection(). Returns the consulting TerminalConnection that was used to create this CiscoConsultCall. You can use this method to correlate a consultation call with its original call. The original call does not have any methods that you can use to determine the consultation call, if any, to which it is related. Returns: The consulting TerminalConnection of the call that created the call that is referenced by this event

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Call, CallObserver, CallActiveEv and Constant Field Values, on page 1661 for more information.

CiscoEv

The CiscoEv interface extends this code JTAPI javax.telephony.events.Ev interface and serves as the base interface for all Cisco-extended JTAPI events. Every event in this package extends this interface, directly or indirectly.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.Ev

Subinterfaces

CiscoAddrActivatedEv, CiscoAddrActivatedOnTerminalEv, CiscoAddrAddedToTerminalEv, CiscoAddrAutoAcceptStatusChangedEv, CiscoAddrCreatedEv, CiscoAddrEv, CiscoAddrInServiceEv, CiscoAddrIntercomInfoChangedEv, CiscoAddrIntercomInfoRestorationFailedEv, CiscoAddrOutOfServiceEv, CiscoAddrRecordingConfigChangedEv, CiscoAddrRemovedEv, CiscoAddrRemovedFromTerminalEv, CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv, CiscoCallChangedEv, CiscoCallEv, CiscoCallSecurityStatusChangedEv, CiscoConferenceChainAddedEv, CiscoConferenceChainRemovedEv, CiscoConferenceEndEv, CiscoConferenceStartEv, CiscoConsultCallActiveEv, CiscoMediaOpenLogicalChannelEv, CiscoOutOfServiceEv, CiscoProvCallParkEv, CiscoProvEv, CiscoProvFeatureEv, CiscoProvTerminalCapabilityChangedEv, CiscoRestrictedEv, CiscoRTPInputKeyEv, CiscoRTPInputStartedEv, CiscoRTPInputStoppedEv, CiscoRTPOutputKeyEv, CiscoRTPOutputStartedEv, CiscoRTPOutputStoppedEv, CiscoTermActivatedEv, CiscoTermButtonPressedEv, CiscoTermCreatedEv, CiscoTermDataEv, CiscoTermDeviceStateActiveEv, CiscoTermDeviceStateAlertingEv, CiscoTermDeviceStateHeldEv, CiscoTermDeviceStateIdleEv, CiscoTermDeviceStateWhisperEv, CiscoTermDNDOptionChangedEv, CiscoTermDNDStatusChangedEv, CiscoTermEv, CiscoTermInServiceEv, CiscoTermOutOfServiceEv, CiscoTermRegistrationFailedEv, CiscoTermRemovedEv, CiscoTermRestrictedEv, CiscoTermSnapshotCompletedEv, CiscoTermSnapshotEv, CiscoToneChangedEv, CiscoTransferEndEv, CiscoTransferStartEv

Declaration

public interface CiscoEv extends javax.telephony.events.Ev

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Ev for more information.

CiscoFeatureReason

The CiscoFeatureReason interface specifies the feature reason that is associated with each delivered event.

Interface History

Cisco Unified Communications Manager Release	Description
7.1(1 and 2)	Added new reason, FORWARD_NO_RETRIEVE, for the Park Monitoring and Assisted DPark feature.
8.0(1)	A new reason code, REASON_SAF_CCD_PSTN_FAILOVER, has been added to convey the proper reason for a PSTN failover to the application.
8.5(1)	A new feature reason, REASON_MEDIA_STREAMING, is added.
10.0.1	A new feature reason, REASON_PLAY_ANNOUNCEMENT, is added.

Declaration

public interface CiscoFeatureReason

Fields

Table 102: Fields in CiscoFeatureReason

Interface	Field	Description
static int	REASON_BARGE	Indicates that the reason for the event is BARGE feature.
static int	REASON_BLINDTRANSFER	Indicates that reason is single step transfer
static int	REASON_CALLPICKUP	Indicates that the reason for the events is PICKUP
static int	REASON_CCM_REDIRECTION	Indicates that the reason for the events is SIP 3xx feature.
static int	REASON_CLICK_TO_CONFERENCE	Indicates that connections have been added or removed by using the Click to Conference feature
static int	REASON_CONFERENCE	Indicates that the reason for the event is CONFERENCE
static int	REASON_DPARK_CALLPARK	Indicates that the reason for events is DPARK feature
public static final int	REASON_DEQUEUING	Indicates that the event is gererated because the call has got de-queued under the Native Queuing Feature.
public static final int	REASON_DEQUEUING_TIMER_EXPIRED	Indicates that the event is generated because the call is de-queued under the Native Queuing Feature as the maximum queue timer expired.
public static final int	REASON_DEQUEUING_AGENTS_BUSY	Indicates that the event has been gererated because the call has got de-queued under the Native Queuing Feature as the agents were busy and the queue was full.
public static final int	REASON_DEQUEUING_AGENTS_UNAVAILABLE	Indicates that the event is gererated because the call has got de-queued under the Native Queuing Feature as the agents were either not logged-in or were unregistered.
static int	REASON_DPARK_REVERSION	Indicates that the reason for events in DPARK Reversion
static int	REASON_DPARK_UNPARK	Indicates that the reason for events in DPARK UNPARK
static int	REASON_FAC_CMC	Indicates that the reason for the events is FAC, CMC feature
static int	REASON_FORWARDALL	Indicates that reason for the event is FORWARD
static int	REASON_FORWARDBUSY	Indicates that the reasons for the event is forward busy
static int	REASON_FORWARDNOANSWER	Indicates that the reasons for the event is forward no answer

Interface	Field	Description
static int	REASON_FORWARD_NO_RETRIEVE	Indications that the reason for the event is forward no retrieve
static int	REASON_IMMDIVERT	Indicates that the reason for the events is imm divert
static int	REASON_MEDIA_STREAMING	Indicates that the event received is related to an Agent Greeting call
static int	REASON_MOBILITY	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_MOBILITY_CELLPICKUP	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_MOBILITY_FOLLOWME	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_MOBILITY_HANDIN	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_MOBILITY_HANDOUT	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_MOBILITY_IVR	Indicates that the reason for events caused by Mobility Manager feature
static int	REASON_NORMAL	Indicates that the reason for the event is NORMAL
static int	REASON_PARK	Indicates that the reason for the event is PARK feature
static int	REASON_PARKREMAINDER	Indicates that the reasons for the event is park remainder
public static final int	REASON_PLAY_ANNOUNCEMENT	This interface indicates that the event was generated because of a play announcement.
static int	REASON_QSIG_PR	Indicates that the reason for the event is QSIG path replacement
public static final int	REASON_QUEUING	Indicates that the event is generated due to the Native Queuing feature.
static int	REASON_REDIRECT	Indicates that the reason for event is REDIRECT
static int	REASON_REFER	Returned for events sent for REFER done at Cisco Unified Communications Manager
static int	REASON_REPLACE	REASON_REPLACE : This reason will be returned for events send for REPLACE feature done at Cisco Unified Communications Manager
static int	REASON_SILENTMONITORING	Indicates that the reason for events in SILENT MONITORING

Interface	Field	Description
static int	REASON_TRANSFER	Indicates that the reason for the event is TRANSFER
static int	REASON_UNPARK	Indicates that the reason for the event is unpark
static final int	REASON_SAF_CCD_PSTN_FAILOVER	Indicates the reason for PSTN failover to the application

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoHuntConnection

A CiscoHuntConnection in a call indicates that the call is routed through a hunt pilot.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New interface

Declaration

public interface CiscoHuntConnection extends CiscoConnection.

Methods

Table 103: Methods in CiscoHuntConnection

Interface	Method	Description
Connection[]	getAgentConnections()	This method returns an array of connections to the hunt group member or null.

Related Documentation

CiscoIntercomAddress

The CiscoIntercomAddress interface extends the CiscoAddress interface with additional Cisco Unified Communications Manager-specific capabilities for intercom addresses. This interface lets applications initiate intercom calls and take advantage of other intercom-specific features.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.Address, CiscoAddress, CiscoObjectContainer

Declaration

public interface CiscoIntercomAddress extends CiscoAddress

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoAddress

APPLICATION_CONTROLLED_RECORDING, AUTO_RECORDING, AUTOACCEPT_OFF, AUTOACCEPT_ON, AUTOANSWER_OFF, AUTOANSWER_UNKNOWN, AUTOANSWER_WITHHEADSET, AUTOANSWER_WITHSPEAKERSET, EXTERNAL, EXTERNAL_UNKNOWN, IN_SERVICE, INTERNAL, MONITORING_TARGET, NO_RECORDING, OUT_OF_SERVICE, RINGER_DEFAULT, RINGER_DISABLE, RINGER_ENABLE, UNKNOWN

Methods

Table 104: Methods in CiscoIntercomAddress

Interface	Method	Description
void	setIntercomTarget(java. lang. String targetDNjava. lang. String targetAsciiLabel, java. lang. String targetUnicodeLabel)	Sets the intercom target DN, intercom target label, and intercom target Unicode label that appears next to the intercom line on the phone. The phone displays the Unicode label if the phone has that capability; otherwise, the phone displays the ASCII target label. **Throws** javax. telephony. InvalidPartyException means that the target DN is invalid. javax. telephony. InvalidStateException means that the address, terminal, or provider are not in service. **Parameters** • targetDN—Destination DN for the intercom call • targetAsciiLabel—ASCII display label shown next to the intercom line on the phone target • UnicodeLabel—Unicode display label shown on the phone
Boolean	isIntercomTargetSet()	Returns true if an application has overridden the current value, or false if the current value matches the default value configured in the database.
void	resetIntercomTarget()	Resets the intercom target DN, intercom target label, and intercom target Unicode label to their default values. **Throws** javax. telephony. InvalidPartyException javax. telephony. InvalidStateException
java. lang. String	getIntercomTargetNumber()	Returns the current intercom target DN that the application set. If the application has not set the intercom target DN, this interface returns the default intercom target DN that is configured in Cisco Unified Communications Manager Administration. Returns: The intercom target DN number, as a string.
java. lang. String	getIntercomTargetAsciiLabel()	Returns the current intercom target label that the application set. If the application has not set the intercom target label, this interface returns the default intercom target label that is configured in Cisco Unified Communications Manager Administration. Returns: The intercom target label string.

Interface	Method	Description
java. lang. String	getIntercomTargetUnicodeLabel()	Returns the current intercom target Unicode label that the application set. If the application has not set the Unicode label, this interface returns the default intercom target Unicode label that is configured in Cisco Unified Communications Manager Administration. Returns: The intercom Unicode target label string.
java. lang. String	getDefaultIntercomTargetNumber()	Returns the default intercom target DN that is configured through Cisco Unified Communications Manager Administration. Returns: The default intercom target DN number, as a string.
java. lang. String	getDefaultIntercomTargetAsciiLabel()	Returns the default intercom target label that is configured through Cisco Unified Communications Manager Administration. Returns: The default intercom target label string.
java. lang. String	getDefaultIntercomTargetUnicodeLabel()	Returns the default intercom target label that is configured through Cisco Unified Communications Manager Administration. Returns: The default unicode intercom target label string.
javax. telephony. Connection[]	connectIntercom(javax. telephony. Terminal terminal, java. lang. String targetNumber)	Places an intercom call from an originating intercom address to a destination intercom address. Returns:A connection list for the calling and called intercom addresses. **Throws** javax. telephony. InvalidPartyException—The target DN is not a valid number. javax. telephony. InvalidArgumentException—The address is not a CiscoIntercomAddress or the terminal is not a Terminal. javax. telephony. InvalidStateException—The address, terminal, or provider is not in service. javax. telephony. ResourceUnavailableException—A resource is not available to complete the operation. javax. telephony. PrivilegeViolationException—The application does not have sufficient privileges to execute this operation.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoAddress

clearCallConnections, getAddressCallInfo, getAutoAcceptStatus, getAutoAnswerStatus, getInServiceAddrTerminals, getPartition, getRecordingConfig, getRegistrationState,

getRestrictedAddrTerminals, getState, getType, isRestricted, setAutoAcceptStatus, setMessageWaiting, setRingerStatus

From Interface javax.telephony.Address

addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

See CiscoAddress for additional information.

CiscoIsacMediaCapability

The CiscoIsacMediaCapability object specifies the properties for a iSAC encoded RTP stream. Applications that support iSAC media termination use this object when registering a CiscoMediaTerminal.

The packet size and bit rate are variable.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	Interface added in this release for iSac codec which can be used by application to register a CiscoMediaTerminal or CiscoRouteTerminal if they want to use this new MediaCapability.

Superinterfaces

None

Declaration

public class CiscoIsacMediaCapabilityextends CiscoMediaCapability

Constuctors

Table 105: Constructor in CiscoIsacMediaCapability

Interface	Constructor	Description
public	CiscoIsacMediaCapability()	Constructs a CiscoIsacMediaCapability

Fields

None

Inherited Fields

From Class com.cisco.jtapi.extensions.CiscoMediaCapability

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, ISAC, WIDEBAND_256K_10_MILLISECONDS

Methods

None

Inherited Methods

Inherited From Class com.cisco.jtapi.extensions.CiscoMediaCapability

getMaxFramesPerPacket, getPayloadType, isSupported, toString

Inherited From Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

CiscoJtapiException

The CiscoJtapiException interface defines error codes that CTI requests may return. Cisco JTAPI extends all of the JTAPI exceptions to implement this interface. You can get the error codes by casting the exception to CiscoJtapiException and calling the method getErrorCode().

For example, if "e" is any exception caught by an application, the following code checks whether the exception is an instance of CiscoJtapiException.

```
try {
   // some code here
 }
catch ( Exception e ) {
   if( e instanceof CiscoJtapiException){
     CiscoJtapiException ce  =
com.cisco.cti.client.CTIFAILURE.(CiscoJtapiException) e
     int errorCode  =  com.cisco.cti.client.CTIFAILURE.ce.getErrorCode()
                     //returns the ErrorCode.
   }
 }
```

Interface History

Cisco Unified Communications Manager Release Number	Description
7.0	Added this interface.
7.1(1 and 2)	Added new error codes for the Logical Partition feature called: CiscoJtapiException.CTIERR_REDIRECT_CALL_PARTITIONING_POLICY and CiscoJtapiException.CTIERR_FEATURE_NOT_AVAILABLE.
8.0(1)	A new error code is added which will be exposed when the monitoring/recording request is rejected when the supervisor/recorder does not meet the security capabilities of the agent. New error code, OPERATION_NOT_AVAILABLE_IN_CURRENT_STATE, has also been added.
8.5(1)	The following new error codes are added:CTIERR_MEDIA_CONNECTION_FAILED, CTIERR_REQUEST_ALREADY_PENDING, CTIERR_START_STREAM_MEDIA_FAILED, CTIERR_STOP_STREAM_MEDIA_FAILED, CTIERR_NO_STREAMING_MEDIA_SESSION, CTIERR_EXISTING_STREAMING_MEDIA_SESSIONCTIERR_ MEDIA_ALREADY_TERMINATED_STATIC_GETPORT_SUPPORT, CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC_GETPORT_SUPPORT, CTIERR_SSO_DISABLED, CTIERR_SSO_AUTH_SERVER_DOWN.
9.0(1)	The following new error codes are added:CTIERR_INVALID_REMOTE_DESTINATION_NUMBER CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER CTIERR_REMOTEDESTINATION_LIMIT_EXCEEDED CTIERR_REMOTE_DEVICE_REQUEST_FAILED_ACTIVE_RD_NOT_SET CTIERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE CTIERR_DEVICE_ALREADY_REGISTERED_NONEXTEND CTIERR_MEDIA_ALREADY_TERMINATED_EXTEND CTIERR_INVALID_REMOTE_DESTINATION_NAME CTIERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING

Cisco Unified Communications Manager Release Number	Description
10.0(1)	The following new error codes are added: CTIERR_AUTHENTICATION_TYPE_ON_UNSUPPORTED_PORT CTIERR_NO_PERSISTENT_CALL_EXISTS CTIERR_ANNOUNCEMENT_ALREADY_IN_PROGRESS CTIERR_ERROR_PLAYING_ANNOUNCEMENT CTIERR_PLAY_ANNOUNCEMENT_FAILED CTIERR_EXTEND_AND_CONNECT_DESTINATION_NOT_REACHABLE CTIERR_CREATE_PERSISTENT_CALL_FAILED CTIERR_PERSISTENT_CALL_EXISTS CTIERR_OPERATION_NOT_ALLOWED_ON_PERSISTENT_CALL CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_CALL_ACTIVE CTIERR_PERSISTENT_CALL_BEING_SETUP CTIERR_INVALID_SSO_TOKEN_SIZE

Declaration

public interface CiscoJtapiException

Fields

Table 106: Fields in CiscoJtapiException

Interface	Field	Description
static int	ASSOCIATED_LINE_NOT_OPEN	This error indicates that the request is issued on a line, which is not open
static int	CALL_ALREADY_EXISTS	This error indicates that another call already exists on the line
static int	CALL_DROPPED	The call dropped after the feature request (hold, unhold, transfer, or conference) but before the request was completed.
static int	CALLHANDLE_NOTINCOMINGCALL	This error indicates that an attempt is made to answer a call that either does not exist or is not in the correct state
static int	CALLHANDLE_UNKNOWN_TO_ LINECONTROL	This error indicates that attempt to redirect call that was unknown to line control

Interface	Field	Description
static int	CANNOT_OPEN_DEVICE	This error indicates that device open failed because the associated device is unregistering
static int	CANNOT_TERMINATE_MEDIA_ON_PHONE	This error indicates that media cannot be terminated by an application when the device is a physical phone (the phone always terminates the media)
static int	CFWDALL_ALREADY_SET	This error indicates that attempt to set CFWALL while it is already set
static int	CFWDALL_DESTN_INVALID	This error indicates that attempt to CFWALL to an invalid destination
static int	CLUSTER_LINK_FAILURE	This error indicates that link to one of the cisco unified communications managers failed in the cluster (network error)
static int	COMMAND_NOT_IMPLEMENTED_ON_DEVICE	This error indicates that device does not support the command.
static int	CONFERENCE_ALREADY_PRESENT	This error indicates that attempt to conference a party that is already in conference
static int	CONFERENCE_FAILED	This error indicates that conference completion was not successful.
static int	CONFERENCE_FULL	This error indicates that all conference bridges are busy.
static int	CONFERENCE_INACTIVE	This error indicates that attempt to complete conference while consult conference is not active
static int	CONFERENCE_INVALID_PARTICIPANT	This error indicates that an attempt to conference to self or an invalid participant
static int	CTIERR_ACCESS_TO_DEVICE_DENIED	This error indicates that the access to device is denied.
public static final int	CTIERR_ANNOUNCEMENT_ALREADY_IN_PROGRESS	This error indicates that the announcement is already in progress.
static int	CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	This error indicates that the application softkeys are already controlled by another application
static int	CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	This error indicates that application data size has exceeded limit
static int	CTIERR_AUTHENTICATION_TYPE_ON_UNSUPPORTED_PORT	This error indicates that the port does not support the authentication type that is used to create the provider.
static int	CTIERR_BIB_NOT_CONFIGURED	This error indicates built in bridge is not configured

Interface	Field	Description
static int	CTIERR_BIB_RESOURCE_NOT_ AVAILABLE	This error indicates that built in bridge resource not available
static int	CTIERR_CALL_MANAGER_NOT_ AVAILABLE	This error indicates that Communications Manager is not available currently
static int	CTIERR_CALL_NOT_EXISTED	This error indicates that call does not exist
static int	CTIERR_CALL_PARK_NO_DN	This error indicates no call park DN
static int	CTIERR_CALL_REQUEST_ALREADY_ OUTSTANDING	This error indicates call request already outstanding
static int	CTIERR_CALL_UNPARK_FAILED	This error indicates that call unpark did not succeed
static int	CTIERR_CAPABILITIES_DO_NOT_MATCH	This error indicates that capabilities do not match
static int	CTIERR_CLOSE_DELAY_NOT_ SUPPORTED_WITH_REG_TYPE	This error indicates that the close delay is not supported with this registration type
static int	CTIERR_CONFERENCE_ALREADY_ EXISTED	This error indicates that conference already exists
static int	CTIERR_CONFERENCE_NOT_EXISTED	This error indicates that conference does not exist
static int	CTIERR_CONNECTION_ON_INVALID_PORT	This error indicates application is trying to connect to invalid port
static int	CTIERR_CONSULT_CALL_FAILURE	This error indicates consult call failure
static int	CTIERR_CONSULTCALL_ALREADY_ OUTSTANDING	This error indicates that consult call already outstanding
static int	CTIERR_CRYPTO_CAPABILITY_MISMATCH	This error indicates that device registration failed as device crypto algorithms does not match with current device registration
static int	CTIERR_CTIHANDLER_PROCESS_ CREATION_FAILED	This error indicates that CTIHandler process creation failed
static int	CTIERR_DB_INITIALIZATION_ERROR	This error indicates DB initialization error
static int	CTIERR_DEVICE_ALREADY_OPENED	This error indicates that device is already opened
static int	CTIERR_DEVICE_NOT_OPENED_YET	This error indicates that device is not yet opened
static int	CTIERR_DEVICE_OWNER_ALIVE_ TIMER_STARTED	This error indicates that there is a device registration failure
static int	CTIERR_DEVICE_REGISTRATION_FAILED_ NOT_SUPPORTED_ MEDIATYPE	This error indicates an invalid media type, CTIPort need to be registered with Dynamic media port registation if it has an intercom line
static int	CTIERR_DEVICE_RESTRICTED	This error indicates that the device is restricted

Interface	Field	Description
static int	CTIERR_DEVICE_SHUTTING_DOWN	This error indicates that device is shutting down
static int	CTIERR_DIRECTORY_LOGIN_TIMEOUT	This error indicates that there is a directory login time out
static int	CTIERR_DUPLICATE_CALL_REFERENCE	This error indicates duplicate call reference
static int	CTIERR_DYNREG_IPADDRMODE_ MISMATCH	This indicates registration failure when Cisco Media/Route Terminal is already registered with different Addressing mode.
public static final int	CTIERR_ERROR_PLAYING_	This error indicates that there was an error playing the announcement.
static int	CTIERR_EXISTING_STREAMING_ MEDIA_SESSION	This error occurs if an application attempts to invoke an Agent Greeting API while another request is made and accepted. JTAPI throws InvalidStateException with a description as “There is an existing streaming media session”.
public static final int	CTIERR_EXTEND_AND_CONNECT_ DESTINATION_NOT_REACHABLE	This error occurs if the extend and connect destination is not reachable.
static int	CTIERR_FAC_CMC_REASON_CMC_ INVALID	Client Matter Code (CMC) entered is invalid
static int	CTIERR_FAC_CMC_REASON_CMC_ NEEDED	CMC is required to offer the call
static int	CTIERR_FAC_CMC_REASON_FAC_ CMC_NEEDED	Forced Authorization Code (FAC) and CMC are required to offer call
static int	CTIERR_FAC_CMC_REASON_FAC_INVALID	FAC entered is invalid
static int	CTIERR_FAC_CMC_REASON_FAC_ NEEDED	FAC is required to offer the call
static int	CTIERR_FEATURE_ALREADY_REGISTERED	This error indicates feature already registered
static int	CTIERR_FEATURE_DATA_REJECT	This error indicates feature data reject
static int	CTIERR_FEATURE_SELECT_FAILED	This error indicates that feature select failed
static int	CTIERR_ILLEGAL_DEVICE_TYPE	This error indicates that the device type is illegal
static int	CTIERR_INCOMPATIBLE_AUTOINSTALL_ PROTOCOL_VERSION	This error indicates that auto install protocol version is incompatible
static int	CTIERR_INCORRECT_MEDIA_CAPABILITY	Device registration failed due to incorrect media capability.
static int	CTIERR_INFORMATION_NOT_AVAILABLE	This error indicates that information is not available

Interface	Field	Description
static int	CTIERR_INTERCOM_SPEEDDIAL_ ALREADY_CONFIGURED	This error indicates that intercom target value is already configured, application is trying to make call with Intercom target DN
static int	CTIERR_INTERCOM_SPEEDDIAL_ ALREADY_SET	This error indicates that intercom request failed as intercom target value is already set, application is trying to set again
static int	CTIERR_INTERCOM_SPEEDDIAL_ DESTN_INVALID	This error indicates that intercomm request failed as intercom target value in not in the intercom group
static int	CTIERR_INTERCOM_TALKBACK_ ALREADY_PENDING	This error indicates that intercom talk back request is already pending
static int	CTIERR_INTERCOM_TALKBACK_FAILURE	This error indicates that talkback request failed for some reason.
static int	CTIERR_INTERNAL_FAILURE	This error indicates there is a CTI internal failure
static int	CTIERR_INVALID_CALLID	This error indicates the call ID is invalid
static int	CTIERR_INVALID_DEVICE_NAME	This error indicates that the device name is not valid
static int	CTIERR_INVALID_DTMFDIGITS	Play DTMF request failed because it is an invalid DTMF digit.
static int	CTIERR_INVALID_FILTER_SIZE	This error indicates that filter size is invalid
static int	CTIERR_INVALID_MEDIA_DEVICE	This error indicates that the media device is not valid
static int	CTIERR_INVALID_MEDIA_PARAMETER	This error indicates media parameter is inavlid
static int	CTIERR_INVALID_MEDIA_PROCESS	This error indicates that there is an invalid media process
static int	CTIERR_INVALID_MEDIA_RESOURCE_ID	This error indicates media resource ID is not valid
static int	CTIERR_INVALID_MESSAGE_HEADER_INFO	This error indicates that the header info is not valid
static int	CTIERR_INVALID_MESSAGE_LENGTH	This error indicates that message length is invalid
static int	CTIERR_INVALID_MONITOR_DESTN	This error indicates monitoring request failed due to invalid monitoring destination
static int	CTIERR_INVALID_MONITOR_DN_TYPE	This error indicates an invalid monitor DN type
static int	CTIERR_INVALID_MONITORMODE	This error indicates monitor request failed due to an invalid monitor mode
static int	CTIERR_INVALID_PARAMETER	This error indicates that the parameter is not valid
static int	CTIERR_INVALID_PARK_DN	This error indicates that the DN is an invalid park DN

Interface	Field	Description
static int	CTIERR_INVALID_PARK_REGISTRATION_ HANDLE	This error indicates that the handle is an invalid park registration handle
static int	CTIERR_INVALID_RESOURCE_TYPE	This error indicates an invalid resource type
static int	CTIERR_IPADDRMODE_MISMATCH	This indicates the registration failure due to IP Addressing Mode mismatch.
static int	CTIERR_LINE_OUT_OF_SERVICE	This error indicates that line is out of service.
static int	CTIERR_LINE_RESTRICTED	This error indicates that the line is restricted
static int	CTIERR_MAXCALL_LIMIT_REACHED	This error indicates that maximum call limit has reached
static int	CTIERR_MEDIA_ALREADY_TERMINATED_ DYNAMIC	This error indicates that device registration failed as device is registered with Dynamic media termination
Final static int	CTIERR_MEDIA_ALREADY_TERMINATED_ DYNAMIC_GETPORT_SUPPORT	This error indicates that the application tries to register a terminal, which is already registered with get port support, with a different registration type.
static int	CTIERR_MEDIA_ALREADY_TERMINATED_ NONE	This error indicates that device registration failed as device is already registered with media termination none
static int	CTIERR_MEDIA_ALREADY_TERMINATED_ STATIC	This error indicates that device registration failed as device is registered with Static media termination
Final static int	CTIERR_MEDIA_ALREADY_TERMINATED_ STATIC_GETPORT_SUPPORT	This error indicates that the application tries to register a terminal, which is already registered with get port support, with a different registration type.
static int	CTIERR_MEDIA_CAPABILITY_MISMATCH	This error indicates that device registration failed as media capability of device does not match with current device registration
static int	CTIERR_MEDIA_CONNECTION_FAILED	This error indicates that there is a general failure with the Agent Greeting feature. JTAPI throws InvalidStateException with a description as “The connection to the media has failed”.
static int	CTIERR_MEDIA_RESOURCE_NAME_ SIZE_EXCEEDED	This error indicates that media resource name size has exceeded limit
static int	CTIERR_MEDIAREGISTRATIONTYPE_ DO_NOT_MATCH	This error indicates that media registration types do not match
static int	CTIERR_MESSAGE_TOO_BIG	This error indicates that message is too big
static int	CTIERR_MORE_ACTIVE_CALLS_THAN_ RESERVED	This error indicates that there are more active calls than reserved

Interface	Field	Description
static int	CTIERR_NO_EXISTING_CALLS	This error indicates there are no existing calls
static int	CTIERR_NO_EXISTING_CONFERENCE	This error indicates that there is no existing conference
public static final int	CTIERR_NO_PERSISTENT_CALL_EXISTS	This error indicates that no persistent call exists.
static int	CTIERR_NO_RECORDING_SESSION	This error indicates recording request failed as there is no recording session
static int	CTIERR_NO_RESPONSE_FROM_MP	This error indicates no response from media resource
static int	CTIERR_NO_STREAMING_MEDIA_SESSION	This error occurs if an application attempts to invoke a stop request while there is no existing media stream to stop. JTAPI throws InvalidStateException with a description as “There is no streaming media session active”.
static int	CTIERR_NOT_PRESERVED_CALL	This error indicates that the call is not preserved
static int	CTIERR_OPERATION_FAILED_QUIETCLEAR	This error indicates that feature unavailable for this call due to temporary failure
static int	CTIERR_OPERATION_NOT_ALLOWED	This error indicates that this operation is not allowed
static int	CTIERR_OUT_OF_BANDWIDTH	This error indicates out of bandwidth error
static int	CTIERR_OWNER_NOT_ALIVE	This error indicates a failure during registering the device
static int	CTIERR_PENDING_ACCEPT_OR_ ANSWER_REQUEST	This error indicates that there is a pending accept or answer request
static int	CTIERR_PENDING_START _MONITORING_REQUEST	This error indicates there is a pending start monitoring request
static int	CTIERR_PENDING_START_RECORDING_ REQUEST	This error indicates there is a pending start recording request
static int	CTIERR_PENDING_STOP_RECORDING_ REQUEST	This error indicates there is a pending stop recording request
public static final int	CTIERR_PLAY_ANNOUNCEMENT_FAILED	This error indicates that the play announcement failed.
static int	CTIERR_PRIMARY_CALL_INVALID	This error indicates that primary call in monitoring request in invalid or gone idle
static int	CTIERR_PRIMARY_CALL_STATE_INVALID	This error indicates that primary call in monitoring request is in invalid state
static int	CTIERR_RECORDING_ALREADY_ INPROGRESS	This error indicates recording request failed that recording is already in progress

Interface	Field	Description
static int	CTIERR_RECORDING_CONFIG_NOT_ MATCHING	This error indicates recording configuration does not match
static int	CTIERR_RECORDING_SESSION_INACTIVE	This error indicates recording request failed because recording session is inactive
static int	CTIERR_REDIRECT_UNAUTHORIZED_ COMMAND_USAGE	This error indicates a redirect unauthorized command usage
static int	CTIERR_REGISTER_FEATURE_ ACTIVATION_FAILED	This error indicates that register feature activation failed
static int	CTIERR_REGISTER_FEATURE_APP_ ALREADY_REGISTERED	Register feature application was already registered
static int	CTIERR_REGISTER_FEATURE_PROVIDER_ NOT_REGISTERED	Register feature provider was not registered.
static int	CTIERR_REQUEST_ALREADY_PENDING	This error occurs if an application attempts to invoke an Agent Greeting API while another request is made. JTAPI throws InvalidStateException with a description as “The request was rejected because there is a similar request already pending”.
static int	CTIERR_RESOURCE_NOT_AVAILABLE	This error indicates that resource is not available to fulfil the request
public static final int	CTIERR_SECURITY_CAPABILITY_MISMATCH	This error code is exposed when the monitoring/recording request is rejected when the supervisor/recorder does not meet the security capabilities of the agent.
int	CTIERR_SSO_AUTH_SERVER_DOWN	This error code is returned if authorization server is down.
int	CTIERR_SSO_DISABLED	This error code is returned if the Single Sign-On feature is not enabled on Cisco Unified Communications Manager.
static int	CTIERR_START_MONITORING_FAILED	This error indicates that start monitoring request failed
static int	CTIERR_START_RECORDING_FAILED	This error indicates that start recording request failed
static int	CTIERR_START_STREAM_MEDIA_FAILED	This error occurs if there is a general failure with the Agent Greeting feature, that is not covered by any of the other error codes.JTAPI throws InvalidStateException with a description as “Start streaming media request failed”.

Interface	Field	Description
static int	CTIERR_STOP_STREAM_MEDIA_FAILED	This error occurs if there is a general failure with the Agent Greeting feature, that is not covered by any of the other error codes. JTAPI throws InvalidStateException with a description as “Stop streaming media request failed”.
static int	CTIERR_STATION_SHUT_DOWN	This error indicates that there is a station shutdown
static int	CTIERR_SYSTEM_ERROR	This error indicates CTI system error
static int	CTIERR_UDP_PASS_THROUGH_NOT_ SUPPORTED	This error indicates UDP data passthrough not supported
static int	CTIERR_UNKNOWN_EXCEPTION	This error indicates an unknown exception occured
static int	CTIERR_UNSUPPORTED_CALL_PARK_ TYPE	This error indicates that call park type is not supported
static int	CTIERR_UNSUPPORTED_CFWD_TYPE	This error indicates that the call forward type is unsupported
static int	CTIERR_USER_NOT_AUTH_FOR_SECURITY	This error indicates user is not authorized for secure connection
static int	CTIERR_REDIRECT_CALL_PARTITIONING_ POLICY	This error indicates redirect is not authorized
static int	CTIERR_FEATURE_NOT_AVAILABLE	This error indicates that the feature is unavailable
static int	DARES_INVALID_REQ_TYPE	This error indicates that there is an internal call processing error: DaRes invalid request type
static int	DATA_SIZE_LIMIT_EXCEEDED	This error indicates that XML data object size is bigger than allowed.
static int	DB_ERROR	This error indicates that the device query contained an illegal device type
static int	DB_ILLEGAL_DEVICE_TYPE	This error indicates illegal device type in DB
static int	DB_NO_MORE_DEVICES	This error is no longer used.
static int	DESTINATION_BUSY	This error indicates that destination is busy
static int	DESTINATION_UNKNOWN	This error indicates that destination is not found
static int	DEVICE_ALREADY_REGISTERED	This error indicates that device registration attempt failed, because the device is already registered
static int	DEVICE_NOT_OPEN	This error indicates that an attempt to open a line failed, as the device is not opened or the device is not registered.

Interface	Field	Description
static int	DEVICE_OUT_OF_SERVICE	This error indicates that device is out of service.
static int	DIGIT_GENERATION_ALREADY_IN_ PROGRESS	This error indicates that digit generation is already in progress.
static int	DIGIT_GENERATION_CALLSTATE_ CHANGED	This error indicates that call state is invalid to continue.
static int	DIGIT_GENERATION_WRONG_CALL_ HANDLE	This error indicates that call handle is invalid and call may be gone.
static int	DIGIT_GENERATION_WRONG_CALL_ STATE	This error indicates that call state is not valid to generate digits.
static int	DIRECTORY_LOGIN_FAILED	This error indicates that directory login failed: directory not initialized
static int	DIRECTORY_LOGIN_NOT_ALLOWED	This error indicates that directory login failed
static int	DIRECTORY_TEMPORARY_UNAVAILABLE	This error indicates that directory is temporarily unavailable.
static int	EXISTING_FIRSTPARTY	This error indicates that there is already a device controlling media.
static int	HOLDFAILED	This error indicates that the hold was rejected by line control or call control layers
static int	ILLEGAL_CALLINGPARTY	This error indicates that an attempt was made to originate call using a calling party that is not on the device
static int	ILLEGAL_CALLSTATE	This error indicates line is not in a legal state to invoke the request
static int	ILLEGAL_HANDLE	This error indicates the handle is not valid
static int	ILLEGAL_MESSAGE_FORMAT	This error indicates that there is a QBE protocol error
static int	INCOMPATIBLE_PROTOCOL_VERSION	This error indicates that JTAPI and CTI versions are not compatible : CTI Error Protocol version not supported
static int	INVALID_LINE_HANDLE	This error indicates that attempt to perform a line operation on an invalid line handle.
static int	INVALID_RING_OPTION	This error indicates that the ring option is invalid
static int	LINE_GREATER_THAN_MAX_LINE	This error indicates that line is greater than the maximum available lines on this device
static int	LINE_INFO_DOES_NOT_EXIST	This error indicates that line information does not exist in the database.

Interface	Field	Description
static int	LINE_NOT_PRIMARY	This error indicates that internal error returned from call control.
static int	LINECONTROL_FAILURE	This error indicates line control refuses to allow a new call to be initiated because of its current state.
static int	MAX_NUMBER_OF_CTI_CONNECTIONS_ REACHED	The maximum number of CTI connections was reached.
static int	MSGWAITING_DESTN_INVALID	This error indicates that attempt to set message waiting lamp for an invalid DN; Message Waiting Destination not found.
static int	NO_ACTIVE_DEVICE_FOR_THIRDPARTY	This error indicates there is no active device for thirdparty
static int	NO_CONFERENCE_BRIDGE	This error indicates that no conference bridge available
static int	NOT_INITIALIZED	This error indicates that attempt is made to open a provider before CTI initialization completes
static int	OPERATION_NOT_AVAILABLE_IN_ CURRENT_STATE	This error indicates that the operation is not available in Current state.
static int	PROTOCOL_TIMEOUT	Internal error returned from call control
static int	PROVIDER_ALREADY_OPEN	This error indicates that an attempt is made to reopen provider
static int	PROVIDER_CLOSED	Attempt to close provider while it is already closed
static int	PROVIDER_NOT_OPEN	This error indicates that device list incomplete or device list query timeout or query aborted
static int	REDIRECT_CALL_CALL_TABLE_FULL	This error indicates that internal error is returned from call control
static int	REDIRECT_CALL_DESTINATION_BUSY	This error indicates that the redirect destination is busy
static int	REDIRECT_CALL_DESTINATION_OUT_ OF_ORDER	This error indicates that redirect destination is out of order
static int	REDIRECT_CALL_DIGIT_ANALYSIS_ TIMEOUT	This error indicates a digit analyss time out, this is an internal error returned from call control
static int	REDIRECT_CALL_DOES_NOT_EXIST	This error indicates that an attempt is made to redirect a call that does not exist or is not longer active
static int	REDIRECT_CALL_INCOMPATIBLE_STATE	This error indicates that internal error is returned from call control

Interface	Field	Description
static int	REDIRECT_CALL_MEDIA_CONNECTION_ FAILED	This error indicates media connection failure, this is an internal error returned from call control
static int	REDIRECT_CALL_NORMAL_CLEARING	This error indicates that redirect failed because of normal call clearing
static int	REDIRECT_CALL_ORIGINATOR_ ABANDONED	This error indicates that far end hung up on the call being redirected
static int	REDIRECT_CALL_PARTY_TABLE_ FULL	This error indicates that internal error is returned from call control
static int	REDIRECT_CALL_PENDING_REDIRECT_ TRANSACTION	This error indicates that internal error is returned from call control
static int	REDIRECT_CALL_PROTOCOL_ERROR	This error indicates a protocol error, this is an internal error returned from call control
static int	REDIRECT_CALL_UNKNOWN_DESTINATION	This error indicates that an attempt is made to redirect to an unknown destination
static int	REDIRECT_CALL_UNKNOWN_ERROR	This error indicates that internal error is returned from call control
static int	REDIRECT_CALL_UNKNOWN_PARTY	This error indicates an unknown party is detected, this is an internal error returned from call control
static int	REDIRECT_CALL_UNRECOGNIZED_ MANAGER	This error indicates that internal error is returned from call control
static int	REDIRECT_CALLINFO_ERR	This error indicates that internal error is returned from call control
static int	REDIRECT_ERR	This error indicates that internal error is returned from call control
static int	RETRIEVEFAILED	This error indicates that retrieval of call was rejected by line control or call control
static int	RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	This error indicates that error occurred in retrieving held call; because there is already another active call on the line
static int	SSAPI_NOT_REGISTERED	This error indicates that the redirect command was issued when the internal supporting interface was not initialized; either CTI has not yet finished its initialization or an internal error occurred
static int	TIMEOUT	This error indicates that the request has timed out.
static int	TRANSFER_INACTIVE	This error indicates that attempt to complete transfer, while consult tranfer is not there

Interface	Field	Description
static int	TRANSFERFAILED	This error indicates that the transfer failed probably because one of the call legs was hung up or disconnected from the far end
static int	TRANSFERFAILED_CALLCONTROL_ TIMEOUT	This error indicates that expected response from call control not received during a transfer
static int	TRANSFERFAILED_DESTINATION_BUSY	This error indicates that an attempt is made to transfer call to a busy destination
static int	TRANSFERFAILED_DESTINATION_ UNALLOCATED	This error indicates an attempt is made to to transfer call to a directory number that is not registered
static int	TRANSFERFAILED_OUTSTANDING_ TRANSFER	This error indicates that existing transfer is still in progress
static int	UNDEFINED_LINE	This error indicates that the line that was specified, is not found on the device
static int	UNKNOWN_GLOBAL_CALL_HANDLE	This error indicates that the global call handle is unknown
static int	UNRECOGNIZABLE_PDU	This error indicates that there is a QBE protocol error
static int	UNSPECIFIED	This error indicates that an unspecified error has occured.

Inherited Fields

None

Methods

Table 107: Methods in CiscoJtapiException

Interface	Method	Description
int	getErrorCode()	Returns the errorCode as an integer for this exception.
java.lang.String	getErrorDescription()	Returns the detailed description of the errorCode
java.lang.String	getErrorDescription(int errorCode)	Deprecated Use String getErrorDescription (); instead. Returns the detailed description of the errorCode.
java.lang.String	getErrorName()	Returns an exception in string format.

Interface	Method	Description
java.lang.String	getErrorName(int errorCode)	Deprecated Use String getErrorName (); instead. Returns an exception in string format.

Inherited Methods

None

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoMediaStreamStartedEv

Applications receive the event when they observe a device that is the target of a “addMediaStream()” invocation. This is the Agent device. This event is sent when the media begins to play on the call.

This event is only deliviered to the device that invokes the original request. Multiple observers on the same address receive the events. Shared lines of the invoking device will not receive this event.

Declaration

public interface CiscoMediaStreamStartedEv extends CiscoCallEv

Fields

None

Inherited Fields

None

Methods

None

Inherited Methods

None

CiscoMediaStreamEndedEv

Applications receive the event when they observe a device that is the target of a “addMediaStream()” invocation. This is the Agent device. This event is sent when the media has finished playing on the call. It contains a field that it exposes if the media is played successfully, or if the end event is the result of an error.

This event is only delivered to the device that invokes the original request. Shared lines of the invoking device will not receive this event.

Declaration

public interface CiscoMediaStreamEndedEv extends CiscoCallEv

Fields

Table 108: Fields in CiscoMediaStreamEndedEv

Interface	Field	Description
static int	RESULT_FAILED	This result code indicates that the CiscoMediaStreamEndedEv is received due to some failure with the request that caused it to end early.
static int	RESULT_SUCCESS	This result code indicates that the CiscoMediaStreamEndedEv is received as a result of successful media streaming.
static int	RESULT_PRIMARY_CALL_DROPPED	This result code indicates that the CiscoMediaStreamEndedEv is received due to the primary call.

Inherited Fields

None

Methods

Table 109: Fields in CiscoMediaStreamEndedEv

Interface	Method	Description
boloean	getResult()	Returns one of the above result codes, which allows the applications to figure out if the CiscoMediaStreamEndedEv is received due to an error, or upon a successful request.

Inherited Methods

None

CiscoJtapiPeer

By extending the com.cisco.services.tracing.TraceModule interface, CiscoJtapiPeer exposes trace information to applications. All instances of JtapiPeer objects that the Cisco JTAPI implementation creates implement this interface. Applications that want to manipulate the trace settings of the Cisco JTAPI implementation may use the CiscoJtapiPeer.getTraceManager method to obtain its TraceManager object. Applications can then manipulate the TraceManager object as described in the com.cisco.services.tracing package.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.5(1)	Added a new API getprovider().

Superinterfaces

CiscoObjectContainer, javax.telephony.JtapiPeer, TraceModule

Declaration

public interface CiscoJtapiPeer extends TraceModule, javax.telephony.JtapiPeer, CiscoObjectContainer

Fields

None

Methods

Table 110: Methods in CiscoJtapiPeer

Interface	Method	Description
CiscoJtapiProperties	getJtapiProperties ()	Defines the various methods that applications can use to modify the parameters that the JTAPI layer will use.
void	setJtapiProperties (CiscoJtapiProperties jtapiproperties)	Provides applications ability to save changes made to CiscoJtapiProperties in jtapi.ini file and activate these changes in properties for the providers in JTAPIPeer.
	getprovider ()	Enhanced to read the singlesignon ticket as ssoticket = "ssoticketfromAD".

Inherited Methods

From Interface com.cisco.services.tracing.TraceModule

getTraceManager, getTraceModuleName

From Interface javax.telephony.JtapiPeer

getName, getProvider, getServices

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

See CiscoJtapiProperties and TraceModule for more information.

CiscoJtapiPeerImpl

This interface has a method called “getProvider(), ” which is the primary way for applications to open a JTAPI provider. This method takes a “provider string, ” and it was enhanced to take a new argument.

Declaration

public interface CiscoJtapiPeerImpl extends ObjectContainerImpl implements CiscoJtapiPeer

Fields

Methods

Table 111: Methods in CiscoJtapiPeerImpl

Interface	Method	Description
provider	getProvider(String providerString)	The provider string argument is a string of key or value pairs . A new key was added to this method to allow applications to specify whether to run in FIPS compliant mode. The new argument is “FIPSCompliant, ” and applications should specify “true” or “false.” Specifying any value for the FIPSCompliant parameter in the Provider String will have no affect if the provider is not configured as a secured connection.

CiscoJtapiProperties

Cisco Unified JTAPI behavior and functionality is tailored by many parameters which are read in from the jtapi.ini file when an instance of CiscoJtapiPeer is instantiated. These parameters are now exposed to applications for control via this CiscoJtapiproperties interface.

Applications can query the CiscoJtapiproperties properties object and change these parameters to better suit the application functionality. Exposing these properties via the CiscoJtapiproperties interface also allows applications to have a single point of administration (at the application end) for these parameters. The most visible parameters are those describing the tracing levels and tracing destinations.

Usage

```
JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
if(peer instanceof CiscoJtapiPeer)
  {
 CiscoJtapiProperties jProps  =  ((CiscoJtapiPeer)peer).getJtapiProperties();
 jProps.setTracePath("\\D:\\Traces\\WorkFlow");
 jProps.setUseJavaConsoleTrace(false);
 MyProviderObserver providerObserver  =  new MyProviderObserver ();
 provider  =  peer.getProvider ( providerName );
 }
```

In the above example an application has set the Java console tracing to off and set the trace path to D:\Traces\WorkFlowApp1.When the peer is obtained an object implementing CiscoJtapiProperties is created by reading parameters set in the jtapi.ini file. If no jtapi.ini file exists in the classpath default settings are used to create this object. The parameters used by Cisco Jtapi are read in and frozen when the first getProvider () call is made.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.0(1)	Enhanced with methods to enable/disable the HuntList feature.
8.6(1)	Enhanced with methods for applications to specify a desired level of FIPS compliance when they download certificates.

Declaration

public interface CiscoJtapiProperties

Fields

None

Sample Code

```
try
 {
  JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
  MyProviderObserver providerObserver  =  new MyProviderObserver ();
  provider  =  peer.getProvider ( ipaddress;login = useid;passwd = password );
  if ( provider ! =  null )
  {
   provider.addObserver ( providerObserver );
   provInService.waitTrue();
   boolean hlenabled  =
((CiscoJtapiPeer)peer).getJtapiProperties().getHuntListEnabled();
   System.out.Println("Initial state of HuntList  =  " + hlenabled);
   CiscoJtapiProperties cjp  =  ((CiscoJtapiPeer)peer).getJtapiProperties();
   cjp.setHuntListEnabled(true);
   ((CiscoJtapiPeer)peer).setJtapiProperties(cjp);
   boolean hlenabled  =
((CiscoJtapiPeer)peer).getJtapiProperties().getHuntListEnabled();
   System.out.Println("Final state of HuntList  =  " + hlenabled);
  }
```

Methods

Table 112: Methods in CiscoJtapiProperties

Interface	Method	Description
void	deleteCertificates(java.lang.String username, java.lang.String instanceID, java.lang.String ccmCAPFAddress, java.lang.String certificatePath)	Deletes X.509 client certificate installed for USER Instance in certificate store.
void	deleteSecurityPropertyForInstance(java.lang.String username, java.lang.String instanceID, java.lang.String capfIp, java.lang.String certPath)	Deletes security properly from jtapi.ini file and also delete certificate previously installed for username/instanceId.
java.lang.String	getAlarmServiceHostname()	Gets the alarm service host name.
int	getAlarmServicePort()	Gets the port number for the alarm service.
boolean	getCallSecurityStatusChangedEv()	Advises the application if it would receive the event CallSecurityStatusChangedEv when applicable.
int	getCtiRequestTimeout()	Gets the timout for cti requests, other than the provider open (seconds).
java.lang.String[]	getDebuggingNames()	Get names of supported debugging level jtapi traces.
boolean	getDebuggingValue(java.lang.String debuggingName)	Get the enabled or disabled state of a debugging level trace.
int	getDesiredServerHeartbeatInterval()	Get the desired interval at which the CTI Manager must send heartbeats to JTAPI (seconds).

Interface	Method	Description
boolean	getDiscConnBeforeCreatingInCPIC()	Controls the event order for the scenario when only redirected party in observed by Application This interface returns True if ConnDisconnectedEv is send before ConnCreatedEv, False otherwise.
java.lang.String	getFileNameBase()	Gets the filename for individual log files.
java.lang.String	getFileNameExtension()	Gets the filename extension for log files.
Boolean	getHuntListEnabled()	Returns true if HuntList is enabled else false.
int	getJavaSocketConnectTimeout()	Returns the value of service parameter for SOCKET CONNECT TIMEOUT in seconds.
int	getNumTraceFiles()	Gets the number of trace files before rollover.
boolean	getPeriodicWakeupEnabled()	Gets the enabled state of periodic wake up.
int	getPeriodicWakeupInterval()	Gets the interval for periodic wakeup (milliseconds).
boolean	getProcessOfferingAfterNewcallevent()	Retrieves the boolean value for the jtapi.ini parameter ProcessOfferringAfterNewcallEvent'. By default this interface returns false.
int	getProviderOpenRequestTimeout()	Gets the timout for a provider open request (seconds).
int	getProviderOpenRetryAttempts()	Returns the value of service parameter for maximum number of reconnect attempts to CTI Manager.
int	getProviderRetryInterval()	Gets the interval at which the connection to the CTI Manager will ge retried (seconds).
int	getQueueSizeThreshold()	Gets the threshold for the event queue size to trigger alarms.
boolean	getQueueStatsEnabled()	Gets the enabled state of event queue stats.
int	getRouteSelectTimeout()	Gets the route select timeout (milliseconds).
java.util.Hashtable	getSecurityPropertyForInstance()	Returns a Hash table with all the parameters set for users and InstanceIDs. See User/InstanceID Hash Table, on page 435 for key and value pairs.
java.util.Hashtable	getSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID)	Teturn a Hash table with all the parameters set for users and InstanceIDs. See User/InstanceID Hash Table, on page 435 for key and value pairs.
java.lang.String[]	getServices()	Returns the services that this implementation supports.
java.lang.String	getSyslogCollector()	Gets the syslog collector hostname.

Interface	Method	Description
int	getSyslogCollectorUDPPort()	Gets the syslog collector UDP port.
java.lang.String	getTraceDirectory()	Path directory where trace files will be written.
int	getTraceFileSize()	Trace file size before rollover.
java.lang.String[]	getTraceNames()	Gets the names of supported jtapi traces.
java.lang.String	getTracePath()	Gets the path where the trace files will be located.
boolean	getTraceValue(java.lang.String traceName)	Gets the enabled or disabled state of a trace.
boolean	getUpdateJtapiCalledWithOriginalCalled()	Queries the parameter setting that changes Jtapi behavior on updating called address.
boolean	getUseAlarmService()	gets the enabled/disabled state of the alarm service.
boolean	getUseFileTrace()	Gets the enabled or disabled state of jtapi log file tracing.
boolean	getUseJavaConsoleTrace()	Gets the enabled or disabled state of jtapi console tracing.
boolean	getUseSameDir()	Causes the traces to go to a single directory if UseSameDir is true.
boolean	getUseSyslog()	Gets the enabled or disabled state of syslog tracing.
boolean	IsCertificateUpdated(java.lang.String user, java.lang.String instanceID)	Provides information about where Client and Server certificates are updated for a given user/instanceID or if the Client and Server certificates are not updated.
void	setAlarmServiceHostname(java.lang.String hostname)	Sets the alarm service host name.
void	setAlarmServicePort(int portNumber)	Sets the port number the alarm service is listening on.
void	setCallSecurityStatusChangedEv(boolean val)	Enables applications to set the filter to receive CallSecurityStatusChangedEv to true or false.
void	setCtiRequestTimeout(int seconds)	Sets the time out for CTI requests other than provider open (seconds).
void	setDebuggingValue(java.lang.String debuggingName, boolean value)	Enables or disables a particular debugging level trace.
void	setDesiredServerHeartbeatInterval(int seconds)	Sets the desired interval at which the CTI Manager must send heartbeats to JTAPI (seconds).
void	setDiscConnBeforeCreatingInCPIC(boolean val)	Sets event order, sent Disconnect before Connection created during redirect at redirted party.
void	setFileNameBase(java.lang.String base)	Sets the filename for log files.
void	setFileNameExtension(java.lang.String extn)	Sets the filename extension for log files.

Interface	Method	Description
void	setHuntListEnabled (boolean)	Enables the Hunt List Feature in Cisco Unified JTAPI.
void	setJavaSocketConnectTimeout(int timeout)	Allows application to set the SOCKET CONNECT TIMEOUT in seconds.
void	setNumTraceFiles(int val)	Sets the number of trace files before rollover.
void	setPeriodicWakeupEnabled(boolean enabled)	Sets the enable/disable state for periodic wake up.
void	setPeriodicWakeupInterval(int milliseconds)	Sets the periodic wake up interval (milliseconds).
void	setProcessOfferingAfterNewcallevent(boolean val)	Controls the event order for the transfer scenario when only transfer destination observed by Application and transfer is completed in offering state.
void	setProviderOpenRequestTimeout(int seconds)	Sets the timeout for a provider open request (seconds).
void	setProviderOpenRetryAttempts(int retryAttempts)	Allows application to set the JTAPI Reconnect Attempts to CTI Manager.
void	setProviderRetryInterval(int seconds)	Sets the interval at which the connection to the CTI Manager will ge retried (seconds).
void	setQueueSizeThreshold(int size)	Sets the threshold for the event queue size to trigger alarms.
void	setQueueStatsEnabled(boolean enabled)	Enables and disables event queue statistics.
void	setRouteSelectTimeout(int milliseconds)	Sets the route select timeout milliseconds.
void	setSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID, java.lang.String authCode, java.lang.String tftp, java.lang.String tftpPort, java.lang.String capf, java.lang.String capfPort, java.lang.String certPath, boolean securityOption)	Deprecated This method is replaced by overloaded method setSecurityPropertyForInstance which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.
void	setSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID, java.lang.String authCode, java.lang.String tftp, java.lang.String tftpPort, java.lang.String capf, java.lang.String capfPort, java.lang.String certPath, boolean securityOption, java.lang.String certstorePassphrase)	Provides the application ability to downloading server/client cerfiticate and set security property for application instance in jtapi.ini file of JTAPI.

Interface	Method	Description
void	setSecurityPropertyForInstance(String user, String instanceID, String authCode, String tftp, String tftpPort, String capf, String capfPort, String certPath, boolean securityOption, String certstorePassphrase, boolean fipsCompliant)	This method provides applications the ability to download server/client certificate and set the security property for this application instance in the jtapi.ini file. Specifying any value of fipsCompliant for this method will have no effect unless the securityOption is set to true. It should be noted that this method will call updateCertificate() and updateServerCertificate(). This is the preferred way to acquire certificates.
void	setServices(java.lang.String[] services)	Sets a list of available services.
void	setSyslogCollector(java.lang.String value)	Sets the syslog collector hostname.
void	setSyslogCollectorUDPPort(int port)	Sets the syslog collector UDP port.
void	setTraceDirectory(java.lang.String dir)	Sets the directory where jtapi trace files should be written.
void	setTraceFileSize(int val)	Sets the size of the trace file.
void	setTracePath(java.lang.String path)	Sets the directory root where jtapi traces will be written.
void	setTraceValue(java.lang.String traceName, boolean value)	Enables or disables a particular trace.
void	setUpdateJtapiCalledWithOriginalCalled(boolean val)	Updates Jtapi Called information with original called once the parameter is set to true always.
void	setUseAlarmService(boolean value)	Enables or disables the alarm service.
void	setUseFileTrace(boolean value)	Enables or disables jtapi log file tracing.
void	setUseJavaConsoleTrace(boolean value)	Enables or disables jtapi console tracing.
void	setUseSameDir(boolean value)	Causes the traces to go to a single directory if UseSameDir is true.
void	setUseSyslog(boolean value)	Enables or disables syslog tracing.
void	updateCertificate(java.lang.String user, java.lang.String intanceID, java.lang.String authcode, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath)	Deprecated This method is replaced by overloaded method updateCertifcate which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.

Interface	Method	Description
void	updateCertificate(java.lang.String user, java.lang.String intanceID, java.lang.String authcode, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath, java.lang.String certStorePassphrase)	Installs an X.509 client certificate for USER Instance in cerfiticate store.
void	updateCertificate(String user, String intanceID, String authcode, String ccmTFTPAddress, String ccmTFTPPort, String ccmCAPFAddress, String ccmCAPFPort, String certificatePath, String certStorePassphrase, boolean fipsCompliant) throws Exception, IOException, UnknownHostException;	This method downloads the client and server certificates for the specified parameters.
void	updateServerCertificate(java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath)	Deprecated This method is replaced by overloaded method updateServerCertifcate which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.
void	updateServerCertificate(java.lang.String userName, java.lang.String instanceID, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath, java.lang.String certStorePassphrase)	Installs an X.509 server certificate given certificate path.
void	updateServerCertificate(String userName, String instanceID, String ccmTFTPAddress, String ccmTFTPPort, String ccmCAPFAddress, String ccmCAPFPort, String certificatePath, String certStorePassphrase, boolean fipsCompliant) throws Exception, IOException, UnknownHostException;	This method downloads the server certificates for the specified parameters. It is called by updateCertificate().

User/InstanceID Hash Table

Table 113: User/InstanceID Hash Table

Key	Value
“user”	userName
String "instanceID"	InstanceID
String"AuthCode"	authCode
String "CAPF"	capfServerIP-Address
String "CAPFPort"	capfServer IP-Address port

Key	Value
String "TFTP"	tftpServer IP-Address
String "TFTPPort"	tftpServer IP-Address port
String "CertPath"	certificate Path
String "securityOption"	Boolean security option (true for enable/ false for disabled)
String "certificateStatus"	Boolean certificate status (true for updated/ false for not updated)

Related Documentation

CiscoLocales

This interface lists all the locales that Cisco Unified JTAPI supports.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoLocales

Fields

Table 114: Fields in CiscoLocales

Interface	Field
static int	LOCALE_ARABIC_ALGERIA
static int	LOCALE_ARABIC_BAHRAIN
static int	LOCALE_ARABIC_EGYPT
static int	LOCALE_ARABIC_IRAQ
static int	LOCALE_ARABIC_JORDAN
static int	LOCALE_ARABIC_KUWAIT
static int	LOCALE_ARABIC_LEBANON

Interface	Field
static int	LOCALE_ARABIC_MOROCCO
static int	LOCALE_ARABIC_OMAN
static int	LOCALE_ARABIC_QATAR
static int	LOCALE_ARABIC_SAUDI_ARABIA
static int	LOCALE_ARABIC_TUNISIA
static int	LOCALE_ARABIC_UNITED_ARAB_EMIRATES
static int	LOCALE_ARABIC_YEMEN
static int	LOCALE_BULGARIAN_BULGARIA
static int	LOCALE_CATALAN_SPAIN
static int	LOCALE_CHINESE_HONG_KONG
static int	LOCALE_CROATIAN_CROATIA
static int	LOCALE_CZECH_CZECH_REPUBLIC
static int	LOCALE_DANISH_DENMARK
static int	LOCALE_DUTCH_NETHERLAND
static int	LOCALE_ENGLISH_UNITED_KINGDOM
static int	LOCALE_ENGLISH_UNITED_STATES
static int	LOCALE_FINNISH_FINLAND
static int	LOCALE_FRENCH_FRANCE
static int	LOCALE_GERMAN_GERMANY
static int	LOCALE_GREEK_GREECE
static int	LOCALE_HEBREW_ISRAEL
static int	LOCALE_HUNGARIAN_HUNGARY
static int	LOCALE_ITALIAN_ITALY
static int	LOCALE_JAPANESE_JAPAN
static int	LOCALE_KOREAN_KOREA
static int	LOCALE_NORWEGIAN_NORWAY
static int	LOCALE_POLISH_POLAND
static int	LOCALE_PORTUGUESE_BRAZIL

Interface	Field
static int	LOCALE_PORTUGUESE_PORTUGAL
static int	LOCALE_ROMANIAN_ROMANIA
static int	LOCALE_RUSSIAN_RUSSIA
static int	LOCALE_SERBIAN_REPUBLIC_OF_MONTENEGRO
static int	LOCALE_SERBIAN_REPUBLIC_OF_SERBIA
static int	LOCALE_SIMPLIFIED_CHINESE_CHINA
static int	LOCALE_SLOVAK_SLOVAKIA
static int	LOCALE_SLOVENIAN_SLOVENIA
static int	LOCALE_SPANISH_SPAIN
static int	LOCALE_SWEDISH_SWEDEN
static int	LOCALE_THAI_THAILAND
static int	LOCALE_TRADITIONAL_CHINESE_CHINA

Methods

None

Related Documentation

See Constant Field Values, on page 1661.

CiscoMasterKeyIndicator

This interface lists the constants for Master Key Indicator.

Table 115: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMasterKeyIndicator

Methods

Table 116: Methods in CiscoMonitorInitiatorInfo

Interface	Method	Description
static final int	NOT_AVAILABLE	Indicates that MKI (Master Key Indicator) is not present.
static final int	AVAILABLE	Indicates that MKI (Master Key Indicator) is present.

CiscoMediaConnectionMode

The CiscoMediaConnectionMode interface lists all of the media connection modes.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoMediaConnectionMode

Fields

Table 117: Fields in CiscoMediaConnectionMode

Interface	Field	Description
static int	NONE	There is no active transmit or receive channel.
static int	RECEIVE_ONLY	Only the receive channel is active.
static int	TRANSMIT_AND_RECEIVE	Both the transmit and the receive channels are active.
static int	TRANSMIT_ONLY	Only the transmit channel is active.

Methods

None

Related Documentation

See Constant Field Values, on page 1661.

CiscoMediaEncryptionAlgorithmType

The CiscoMediaEncryptionAlgorithmType interface indicates the SRTP algorithm type used for encryption. This interface lists all of the security indicator values that an application can get in CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv. If an application is terminating its own media on CTIPorts and Media Terminated RPs, only one of the following algorithms needs to be provided in the register API.

Interface History

Cisco Unified Communications Manager Release	Description
3.x	Added the extension.

Superinterfaces

public interface CiscoMediaEncryptionAlgorithmType

Fields

Table 118: Fields in CiscoMediaEncryptionAlgorithmType

Inteface	Field	Description
staticint	AES_128_COUNTER	The algorithm used is based on Advanced Encryption Standard (AES), which is a computer security standard. The cryptography scheme is a symmetric block cipher that encrypts and decrypts 128-bit blocks of data.

Related Documentation

See Constant Field Values, on page 1661 for additional information.

CiscoMediaEncryptionKeyInfo

The CiscoMediaEncryptionKeyInfo interface lets applications get information about SRTP keys.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoMediaEncryptionKeyInfo

Fields

None

Methods

Table 119: Methods in CiscoMediaEncryptionKeyInfo

Interface	Method	Description
int	getAlgorithmID()	Returns the media encryption algorithm ID for the current stream.
int	getIsMKIPresent()	Indicates whether MKI is present.
byte[]	getKey()	Returns the master key for the stream.
int	getKeyLength()	Returns the keyLength of the key.
byte[]	getSalt()	Returns the salt key for the stream.
int	getSaltLength()	Returns the saltLength of the salt.
int	keyDerivationRate()	Indicates the SRTP key derivation rate for this session.

Related Documentation

See CiscoRTPInputKeyEv, CiscoRTPOutputKeyEv.

CiscoMediaOpenIPPortEv

CiscoMediaOpenIPPortEv event is delivered only if the terminal is registered with registration type as CiscoBaseTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT or CiscoBaseTerminal.STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.5(1)	New interface.

Sample Code

```
public class MyTermObserver implements implements TerminalObserver,
    CallControlTerminalObserver,  AgentTerminalObserver,  PhoneTerminalObserver
 {
  public void termChangedEvent (TermEv[] evlist)
   {
   for(int i = 0; evlist ! =  null && i < evlist.length; i++)
     {
      …
      …
      If ( evlisth[i] instanceof  CiscoMediaOpenIPPortEv)
        {
        CiscoMediaOpenIPPortEv ev  =  (CiscoMediaOpenIPPortEv)evlist[i];
        if(((CiscoBaseMediaTerminal)(ev.getTerminal())))).getRegistrationType
             =  =  CiscoTerminal.DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT)

          {
          System.out.println("Set RTP parameters");
          System.out.println("open the port");
          } else {
          System.out.println("Open port");
          }
        }
      }
        …
        …
  }
```

Declaration

public interface CiscoMediaOpenIPPortEv

Superinterfaces

NA

Fields

NA

Inherited Fields

NA

Methods

Table 120: Methods in CiscoMediaOpenIPPortEv

Interface	Method
int	getMediaIPAddressingMode()
CiscoRTPHandle	getCiscoRTPHandle()

Inherited Methods

NA

CiscoMediaOpenLogicalChannelEv

The system sends a CiscoMediaOpenLogicalChannelEv event each time that media gets established for a dynamically registered CiscoMediaTerminal or CiscoRouteTerminal. Upon receiving this event, applications must invoke setRTPParams on CiscoMediaTerminal or CiscoRouteTerminal and pass in the IP address and port number where they want to terminate the media, along with the rtpHandle that this event delivers.

Applications can get a call reference by using CiscoProvider.getCall(CiscoRTPHandle). Applications must be aware that the far end and local end may not be able to invoke features unless the setRTPParams method is invoked. If applications fail to respond to this event within the specified time, the call may get disconnected.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.0(1)	Added the getAddressingModeForMedia() method.
8.5(1)	Added the isRTPRequired() method.

Sample Code

```
public class MyTermObserver implements implements TerminalObserver,
       CallControlTerminalObserver,  AgentTerminalObserver,  PhoneTerminalObserver

 {
  public void termChangedEvent (TermEv[] evlist)
   {
   for(int i = 0; evlist ! =  null && i < evlist.length; i++){
     …
     …
     If ( evlisth[i] instanceof  CiscoMediaOpenLogicalChannelEv)
      {
      CiscoMediaOpenLogicalChannelEv ev  =
(CiscoMediaOpenLogicalChannelEv)evlist[i];
      if(ev.isRTPRequired())
      {
      System.out.println("Set RTP parameters");
```

```
      } else {
      System.out.println("Do not set RTP parameters");
      }
   }
 }
     …
     …
 }
```

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoMediaOpenLogicalChannelEv extends CiscoTermEv

Fields

Table 121: Fields in CiscoMediaOpenLogicalChannelEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 122: Methods in CiscoMediaOpenLogicalChannelEv

Interface	Method	Description
int	getAddressingModeForMedia()	Returns int Application and could get following value for required IP Addressing Mode: • CiscoTerminal.IP_ADDRESSING_IPv4—Means application needs to provide IPv4 format for the IP Address in setRTPParams request. • CiscoTerminal.IP_ADDRESSING_IPv6: Means application need to provide IPv6 format IP Address in set RTP Params request.
CiscoRTPHandle	getCiscoRTPHandle()	Returns CiscoRTPHandle object. Applications should pass this handle along with RTPParameters to CiscoMediaTerminal or CiscoRouteTerminal. Applications can get call reference using CiscoProvider.getCall If there is no callobserver or there was no callobserver when this event is delivered, then CiscoProvider.getCall may return null
int	getMediaConnectionMode()	Returns a CiscoMediaConnectionMode. Applications could get one of the following values: • CiscoMediaConnectionMode. RECEIVE_ONLY—Means one-way media receive only. • CiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE—Means two-way media. Applications should never see a value of NONE; however, if that happens, applications should ignore the event and log an error.
int	getPacketSize()	Returns the packet size of the far end, in milliseconds. getPacketSize

Interface	Method	Description
int	getPayLoadType()	Returns the payload format of the far end, one of the following constants: • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE
boolean	isRTPRequired()	Indicates if the application must set the RTP parameters upon receiving this event.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and CiscoRTPParams

CiscoMediaSecurityIndicator

CiscoMediaSecurityIndicator gets sent in CiscoRTPInputKeyEv, CiscoRTPOutputKeyEv, and CiscoSnapShotRTPEv. It shows the call security status.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoMediaSecurityIndicator

Fields

Table 123: Fields in CiscoMediaSecurityIndicator

Interface	Field	Description
staticint	MEDIA_ENCRYPT_KEYS_AVAILABLE	Terminates the media in secure mode, and keys are available.
staticint	MEDIA_ENCRYPT_KEYS_UNAVAILABLE	Terminates the media in secure mode, but keys are not available because SRTP is not enabled in Cisco Unified Communications Manager Administration.
staticint	MEDIA_ENCRYPT_USER_NOT_AUTHORIZED	Terminates the media in secure mode, but keys are not available because the user is not authorized to get the keys.
staticint	MEDIA_NOT_ENCRYPTED	The media is not encrypted for this call.

Related Documentation

See Constant Field Values, on page 1661.

CiscoMediaTerminal

A CiscoMediaTerminal is a special kind of CiscoTerminal that allows applications to terminate RTP media streams. Unlike a CiscoTerminal, a CiscoMediaTerminal does not represent a physical telephony endpoint, which is observable and controllable in a third-party manner. Instead, a CiscoMediaTerminal is a logical telephony endpoint, which may be associated with any application that wants to terminate media. Such applications include voice messaging systems, interactive voice response (IVR), and softphones.

Note Only CTIPorts appear as CiscoMediaTerminals through Cisco Unified JTAPI.

Terminating media is a two-step process. To terminate media for a particular terminal, an application first adds an observer that implements the CiscoTerminalObserver interface using the Terminal.addObserver method. Finally, the application registers the IP address and port number to which incoming RTP streams for the terminal should be directed, by using the CiscoMediaTerminal.register method.

To supply an IP address and port number dynamically on a per-call basis, applications must register by only providing the capabilities that they support. Applications must react to the CiscoMediaOpenLogicalChannelEv that gets sent whenever media gets established. Applications registering with this type must be aware that, when this event is received, the far end and the local end may not be able to perform any feature operation unless media is established. If applications fail to respond to this event within the specified time, the call may get dropped.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1x	Support added for IPv6.

Superinterfaces

CiscoObjectContainer, CiscoTerminal, javax.telephony.Terminal

Declaration

public interface CiscoMediaTerminal extends CiscoTerminal

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoTerminal

ASCII_ENCODING, DEVICESTATE_ACTIVE, DEVICESTATE_ALERTING, DEVICESTATE_HELD, DEVICESTATE_IDLE, DEVICESTATE_UNKNOWN, DEVICESTATE_WHISPER,

DND_OPTION_CALL_REJECT, DND_OPTION_NONE, DND_OPTION_RINGER_OFF, IN_SERVICE, IP_ADDRESSING_MODE_IPV4, IP_ADDRESSING_MODE_IPV4_V6, IP_ADDRESSING_MODE_IPV6, IP_ADDRESSING_MODE_UNKNOWN, IP_ADDRESSING_MODE_UNKNOWN_ANATRED, NOT_APPLICABLE, OUT_OF_SERVICE, UCS2UNICODE_ENCODING, UNKNOWN_ENCODING

Methods

Table 124: Methods in CiscoMediaTerminal

Interface	Method	Description
void	register(java.net.InetAddressaddress, intport, CiscoMediaCapability[]capabilities)	This method registers the MediaTerminal and returns successfully when the MediaTerminal is registered. The CiscoMediaTerminal must be in the CiscoTerminal.UNREGISTERED state and its Provider must be in the Provider.IN_SERVICE state. This method has three arguments: • The first argument specifies the internet address at which the RTP media stream for this Terminal will terminate. • The second indicates the UDP port at which RTP packets will be directed. • The final argument indicates the type of RTP encodings that the application is willing to support for this Terminal. **Parameters** • address—The internet address at which inbound IPv4 RTP streams on this terminal will terminate • port—The UDP port for inbound RTP streams on this terminal • capabilities—The list of the types of RTP encodings that the application supports for this terminal. **Throws** CiscoRegistrationException

Interface	Method	Description
void	register(java.net.InetAddressaddress, intport)	Deprecated Registers a Terminal with the specified address and port, defaulting to G.711 64 kHz u-law encoding with a thirty-millisecond packet size. **Parameters** • address—The internet address for inbound IPv4 RTP streams on this terminal • port—The UDP port for inbound RTP streams on this terminal **Throws** CiscoRegistrationException
void	register(java.net.InetAddressaddress, intport, CiscoMediaCapability[]capabilities, int[]algorithmIDs)	This method registers the MediaTerminal. Ensure that the CiscoMediaTerminal is in the CiscoTerminal.UNREGISTERED state and its Provider is in the Provider.IN_SERVICE state. This method returns successfully when the MediaTerminal gets registered. This method requires that the application have a TLS link established with CTIManager and have the SRTP Enabled flag enabled in Cisco Unified Communications Manager Administration for the user; otherwise, the system throws a PrivilegeViolationException. **Parameters** • address—The internet address for inbound IPv4 RTP streams on this terminal • port—The UDP port for inbound RTP streams on this terminal • capabilities—The list of RTP encodings that this terminal supports • algorithmIDs—The SRTP algorithms that this CTIPort supports. AlgorithmIDs must be one of CiscoMediaEncryptionAlgorithmType. **Throws** CiscoRegistrationException javax.telephony.PrivilegeViolationException

Interface	Method	Description
void	register(java.net.InetAddressaddress, intport, CiscoMediaCapability[]capabilities, int[]algorithmIDs, java.net.InetAddressaddress_v6, intactiveAddressingMode)	

Interface	Method	Description
		The CiscoMediaTerminal must be in the CiscoTerminal.UNREGISTERED state and its Provider must be in the Provider.IN_SERVICE state. The successful effect of this method is to register the MediaTerminal. The activeAddressingMode indicates the application IP addressing capabilities. If application specifies activeAddressingMode as CiscoTerminal.IP_ADDRESSING_MODE_IPv4, then it must also specify address. If application specifies activeAddressingMode as CiscoTerminal.IP_ADDRESSING_MODE_IPv6, then it must also specify address_v6. If application specifies activeAddressingMode as CiscoTerminal.IP_ADDRESSING_MODE_IPv4_6, then it must also specify address and address_v6. **Method Arguments** This method has four arguments: • The first argument specifies the internet address at which the RTP media stream for this Terminal will be terminated. • The second indicates the UDP port at which RTP packets will be directed. • The third argument indicates the type of RTP encodings that the application is willing to support for this Terminal • The final argument indicates SRTP algorithm that application supports. This method can be used only if application has TLS link established with CTIManager and if application has SRTP Enabled flag enabled in CM Admin pages for the user, otherwise PrivilegeViolationException is thrown. **Method Post-conditions** This method returns successfully when the MediaTerminal is registered. **Parameters** • address—The internet address for inbound IPv4 RTP streams on this terminal, it can be null depending on application Addressing Mode. • port—The UDP port for inbound RTP streams on this terminal

Interface	Method	Description
		• capabilities—The list of RTP encodings supported by this terminal • algorithmIDs—Indicates SRTP algorithms that this CTIPort supports. AlgorithmIDs may only be one of CiscoMediaEncryptionAlgorithmType • address_v6—The IPv6 internet address for inbound IPv6 RTP streams on this terminal, it can be null depending upon activeAddressingMode • activeAddressingMode—IP Addressing mode in which application intends to register this CiscoMediaTerminal. It can be: CiscoTerminal.IP_ADDRESSING_MODE_IPv4 CiscoTerminal.IP_ADDRESSING_MODE_IPv6 CiscoTerminal.IP_ADDRESSING_MODE_IPv4z_v6 **Since: 7.0** **Throws** CiscoRegistrationException, javax.telephony.PrivilegeViolationException

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities)	This method registers the MediaTerminal with the specified CiscoMediaCapabilities. Applications should use this method when they want to supply the IP address and port dynamically for each call. Applications that register with this method will receive a CiscoMediaOpenLogicalChannelEv for each call and must supply an IP address and port number by using the setRTPParams method on this object. Ensure the CiscoMediaTerminal is in the CiscoTerminal.UNREGISTERED state and its Provider is in the Provider.IN_SERVICE state. **Method Arguments** Arguments indicate the type of RTP encodings that the application is willing to support for this Terminal. **Method Post-conditions** This method returns successfully when the CiscoMediaTerminal is registered. **Parameters** capabilities—The list of RTP encodings that this terminal supports. **Throws** CiscoRegistrationException

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities, int[]algorithmIDs)	This method registers a MediaTerminal with the specified CiscoMediaCapabilities and supported SRTP algorithms. Applications should use this method when they want to supply the IP address and port dynamically for each call and also want to specify the SRTP algorithm. Applications that register with this method will receive a CiscoMediaOpenLogicalChannelEv for each call and must supply the IP address and port number by using the setRTPParams method on this object. This form of register() also requires a second parameter that indicates which SRTP algorithm that the application supports. This method requires that the application have a TLS link established with CTIManager and have the SRTP Enabled flag enabled in Cisco Unified Communications Manager Administration for the user; otherwise, the system throws a PrivilegeViolationException. This method returns successfully when the CiscoMediaTerminal gets registered. Ensure the CiscoMediaTerminal is in the CiscoTerminal.UNREGISTERED state and its Provider is in the Provider.IN_SERVICE state. **Parameters** • capabilities—The list of RTP encodings that this terminal supports • algorithmIDs—The list of SRTP algorithms that this terminal supports. AlgorithmIDs must be one of CiscoMediaEncryptionAlgorithmType. **Throws** CiscoRegistrationException javax.telephony.PrivilegeViolationException

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities, int[]algorithmIDs, intactiveAddressingMode)	

Interface	Method	Description
		The CiscoMediaTerminal must be in the CiscoTerminal.UNREGISTERED state and its Provider must be in the Provider.IN_SERVICE state. The successful effect of this method is to register the MediaTerminal. It registers a Terminal with specified CiscoMediaCapabilities and supported SRTP algorithms. It also indicates that application is interested in supplying ipAddress and port dynamically for each call. Applications registering with this method receive CiscoMediaOpenLogicalChannelEv for each call and have to supply ipAddress and port number using setRTPParams method on this object. The second parameter indicates SRTP algorithm that application supports. This method can be used only if application has TLS link established with CTIManager and if application has SRTP Enabled flag enabled in Cisco Unified Communications Manager Administration for the user, otherwise PrivilegeViolationException is thrown. **Method Arguments** Arguments indicate the type of RTP encodings that the application is willing to support for this Terminal and the application or CTIManager failure persistence delay. **Method Post-conditions** This method returns successfully when the CiscoMediaTerminal is registered. **Parameters** • capabilities—The list of RTP encodings supported by this terminal • algorithmIDs—Indicates the list of SRTP algorithms supported by this terminal. AlgorithmIDs may only be one of CiscoMediaEncryptionAlgorithmType • activeAddressingMode—Is the IP Addressing mode in which application intends to register this CiscoMediaTerminal. The activeAddressingMode can be: CiscoTerminal.IP_ADDRESSING_MODE_IPv4 CiscoTerminal.IP_ADDRESSING_MODE_IPv6 CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6 **Throws**

Interface	Method	Description
		CiscoRegistrationException javax.telephony.PrivilegeViolationException
void	setRTPParams(CiscoRTPHandlertpHandle, CiscoRTPParamsrtpParams)	Applications must use this method when they want to set the IP address and RTP port number to dynamically stream media for a call. In this situation, the application will have registered the MediaTerminal or CiscoRouteTeminal by providing only capabilities. Applications must invoke this method upon receiving the CiscoCallOpenLogicalChannel on terminalObserver. Applications must pass in the rtpHandle that they receive in CiscoCallOpenLogicalChannelEv. Applications can get a CiscoCall reference by calling the CiscoProvider.getRTPHandle(rtpHandle) method. This method may return null if no call observer is added on the terminal, or there was no callobserver at the time when this event got sent sent, or or there is no call associated with this handle. **Parameters** • rtpHandle—is obtained from. CiscoMediaCallOpenLogicalChannelEv • rtpParams—is of type CiscoRTPParams, which is used to specify the dynamic RTP address and port number for a media terminal on a per-call basis. **Throws** javax.telephony.InvalidStateException javax.telephony.InvalidArgumentException javax.telephony.PrivilegeViolationException
void	unregister()	This method unregisters the MediaTerminal and returns successfully when the MediaTerminal gets unregistered. The CiscoMediaTerminal must be registered and its Provider must be in the Provider.IN_SERVICE state. **Throws** CiscoUnregistrationException
boolean	isRegistered()	This method returns true if the CiscoMediaTerminal is registered and false otherwise. For a MediaTerminal, this method returns true if the MediaTerminal is InService and false if it is OutOfService. For CTIManager failure cases, this method returns false.

Interface	Method	Description
boolean	isRegisteredByThisApp()	This method returns true if this application issued a successful registration request. The registration remains valid even if the device is out-of-service because of a CTIManager failure. This will get set to true until this application unregisters the device.
int	getIPAddressingMode()	An application can invoke this API to query the IP Addressing Mode of the CiscoMediaTerminal Addressing mode may be any of the following constants: • CiscoTerminal.IP_ADDRESSING_IPv4 • CiscoTerminal.IP_ADDRESSING_IPv6 • CiscoTerminal.IP_ADDRESSING_IPv4_v6

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoTerminal

createSnapshot, getAltScript, getDeviceState, getDNDOption, getDNDStatus, getEMLoginUsername, getFilter, getLocale, getProtocol, getRegistrationState, getRTPInputProperties, getRTPOutputProperties, getState, getSupportedEncoding, isRestricted, sendData, sendData, setDNDStatus, setFilter, unPark

From Interface javax.telephony.Terminal

addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

See CiscoTerminal and CiscoMediaOpenLogicalChannelEv.CiscoRTPParams

CiscoMonitorInitiatorInfo

This interface defines provides information about the monitor initiator.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoMonitorInitiatorInfo

Fields

None

Methods

Table 125: Methods in CiscoMonitorInitiatorInfo

Interface	Method	Description
CiscoAddress	getAddress()	Returns the monitor initiator address.
Int	getMonitorInitiatorCallLegHandle()	Returns the call leg hanlde at the monitor initiator. JTAPI gets the call at the monitor target by using provider.getCall(int monitorInitiatorCallLegHandle). This method returns null if the call at the monitor initiator is not active in this provider.
java.lang.String	getTerminalName()	Returns the terminal name of the monitor initiator.

Related Documentation

None

CiscoMonitorTargetInfo

This interface provides information about the monitor target.

Declaration

public interface CiscoMonitorTargetInfo

Fields

None

Methods

Table 126: Methods in CiscoMonitorTargetInfo

Interface	Method	Description
CiscoAddress	getAddress()	Returns the monitor target address.
Int	getMonitorTargetCallLegHandle()	Returns the call leg handle at the monitor target.
java.lang.String	getTerminalName()	Returns the terminal name of monitor target.

Related Documentation

None

CiscoMultiForkingRecorderInfo

The CiscoMultiForkingRecorderInfo interface contains the information related to the recorders.

Interface History

Cisco Unified Communications Manager Release Number	Description
12.5(1)	A new API which returns recorders details during Multi Forking Recording. `CiscoMultiForkingRecorderInfo()` New constants: CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_UNKNOWN CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_OPTIONAL_RECORDER CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_MANDATORY_RECORDER CALL_RECORDING_MULTI_FORKING_RECORDER_STATUS_UNKNOWN CALL_RECORDING_MULTI_FORKING_RECORDER_STATUS_SUCCESS CALL_RECORDING_MULTI_FORKING_RECORDER_STATUS_FAILURE

Declaration

public interface CiscoMultiForkingRecorderInfo

Methods

Methods in CiscoMultiForkingRecorderInfo

Interface	Method	Description
java.lang.String	getRecorderURI()	This interface returns URI of the MultiForking recorder.
java.lang.String	getRecorderErrorMsg()	This interface returns the error message of the MultiForking recorder.
public int	getRecorderType()	This interface returns the integer which denotes the type of recorder. The recorder type can be: CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_UNKNOWN CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_OPTIONAL_RECORDER CALL_RECORDING_MULTI_FORKING _RECORDER_TYPE_MANDATORY_RECORDER
public int	get RecorderStatus()	This interface returns the integer which denotes the status of the recorder. The recorder status can be: CALL_RECORDING_MULTI_FORKING _RECORDER_STATUS_UNKNOWN CALL_RECORDING_MULTI_FORKING _RECORDER_STATUS_SUCCESS CALL_RECORDING_MULTI_FORKING _RECORDER_STATUS_FAILURE

CiscoMultiMediaCapabilityInfo

CiscoMultiMediaCapabilityInfo interface contains the multimedia capabilities of a terminal. Applications can get the video capability, telepresence interopability, and number of screens information of the terminal using this API.

Declaration

public interface CiscoMultiMediaCapabilityInfo

com.cisco.jtapi.extensions.CiscoMultiMediaCapabilityInfo

Fields

Table 127: Fields in CiscoMultiMediaCapabilityInfo

Interface	Field	Description
static final int	VIDEO_DISABLED	Indicates that the CiscoMultiMediaCapabilityInfo. getVideoCapability () for this terminal is CiscoMultiMediaCapabilityInfo.ENABLED.
static final int	VIDEO_ENABLED	Indicates that the CiscoMultiMediaCapabilityInfo. getVideoCapability () for this terminal is CiscoMultiMediaCapabilityInfo.ENABLED.
static final int	TELEPRESENCEINTEROP_NONE	Indicates that the CiscoMultiMediaCapabilityInfo. getTelepresenceInfo() for this terminal is TELEPRESENCEINTEROP_NONE.
static final int	TELEPRESENCEINTEROP_ENABLED	Indicates that the CiscoMultiMediaCapabilityInfo. getTelepresenceInfo () for this terminal is CiscoMultiMediaCapabilityInfo. TELEPRESENCEINTEROP_ENABLED.
static final int	UNKNOWN	This field will return -1 to indicate that the video capability and telepresence interopability is screen count is Unknown.

Methods

Table 128: Methods in MultiMediaCapabilityInfo

Interface	Method	Description
int	getVideoCapability()	Returns the video capability of the Terminal. The video capability can be CiscoMultiMediaCapabilityInfo. DISABLED or CiscoMultiMediaCapabilityInfo. ENABLED.
int	getTelepresenceInfo()	Returns the telepresence interoperability of the Terminal. The telepresence interoperability can be CiscoMultiMediaCapabilityInfo. TELEPRESENCEINTEROP_DISABLED or CiscoMultiMediaCapabilityInfo. TELEPRESENCEINTEROP_ENABLED.
int	getScreenCount()	Returns the number of screens present on the Terminal.

CiscoMultiMediaConnectionMode

This interface specifies the connection mode associated with the multimedia stream.

Table 129: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMultiMediaConnectionMode

Methods

Table 130: Methods in CiscoProvTerminalMultiMediaConnectionMode

Interface	Field	Description
static final int	RECEIVE_ONLY	Indicates that only the receive channel is active.
static final int	TRANSMIT_ONLY	Indicates that only the transmit channel is active.

CiscoMultiMediaEncryptionKeyInfo

This interface contains the multi media streams encryption key information of a Terminal.

Table 131: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMultiMediaEncryptionKeyInfo

Methods

Table 132: Methods in CiscoProvTerminalMultiMediaEncryptionKeyInfo

Interface	Method	Description
byte[]	getRxKey()	Returns the master key for the input stream.
byte[]	getRxSalt()	Returns the salt key for the input stream.
byte[]	getTxKey()	Returns the master key for the output stream.
byte[]	getTxSalt()	Returns the salt key for the output stream.
int	getAlgorithmID()	Returns the media encryption algorithm ID for the current stream.
int	getRxMKIPresent()	Indicates whether MKI is present for the input stream.
int	getTxMKIPresent()	Indicates whether MKI is present for the output stream.

CiscoMultiMediaProperties

This interface contains the multi media stream information of the video-capabilities on a Terminal.

Table 133: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMultiMediaProperties

Methods

Table 134: Methods in CiscoProvTerminalMultiMediaProperties

Interface	Method	Description
CiscoRTPProperties	CiscoRTPProperties	Returns the rtp properties for the multimedia stream.

Interface	Method	Description
int	getMultiMediaConnection Mode()	Returns the connection mode for the multimedia stream. The multimedia connection mode can be: • CiscoMultiMediaConnectionMode. INACTIVE = 3 • CiscoMultiMediaConnectionMode. RECEIVE_ONLY = 1; • CiscoMultiMediaConnectionMode. TRANSMIT_ONLY = 2 • CiscoMultiMediaConnectionMode. TRANSMIT_AND_RECEIVE = 0
int	getMultiMediaType()	Returns the multimedia type for the multimedia stream. The media type can be • CiscoMultiMediaType. INVALID = 0 • CiscoMultiMediaType. AUDIO = 1 • CiscoMultiMediaType. MAIN_VIDEO = 2 • CiscoMultiMediaType. PRESENTATION_VIDEO = 3
boolean	isKeyInfoPresent()	Returns whether key information is present for the multimedia stream.
CiscoMultiMedia EncryptionKeyInfo	getMultiMediaEncryption KeyInfo()	Returns the multimedia encryption data for the multimedia stream.
int	getMultiMediaSecurity Indicator()	Indicates security indicator for the multimedia stream. The security indicator can be: • CiscoMasterKeyIndicator. NOT_AVAILABLE = 0 • CiscoMasterKeyIndicator. AVAILABLE = 1

CiscoMultiMediaStreamsInfoEv

CiscoMultiMediaStreamsInfoEv is a new interface that is being notified to applications as a Terminal Event. This interface will be delivered to terminal observers added by applications to indicate the multimedia streams information.

Table 135: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMultiMediaStreamsInfoEv extends CiscoTermEv

Methods

Table 136: Methods in CiscoProvTerminalMultiMediaStreamsInfoEv

Interface	Field	Description
CiscoMultiMediaProperties[]	getProperties()	Returns an array of CiscoMultiMediaProperties, one for each stream.
CiscoCallID	getCallID()	Returns CiscoCallID.

CiscoMultiMediaType

This interface specifies the multimedia type associated with the multimedia stream.

Table 137: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoMultiMediaType

Methods

Table 138: Methods in CiscoProvTerminalMultiMediaType

Interface	Field	Description
static final int	INVALID	The multimedia stream type is unknown.

Interface	Field	Description
static final int	AUDIO	The multimedia stream contains audio information.
static final int	MAIN_VIDEO	The multimedia stream contains video information.
static final int	PRESENTATION_VIDEO	The multimedia stream contains presentation information.

CiscoObjectContainer

The ApplicationObject interface allows applications to associate an application-defined object to objects that implement this interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Subinterfaces

CiscoAddress, CiscoCall, CiscoCallID, CiscoConnection, CiscoConnectionID, CiscoConsultCall, CiscoIntercomAddress, CiscoJtapiPeer, CiscoMediaTerminal, CiscoProvider, CiscoRouteTerminal, CiscoTerminal, CiscoTerminalConnection

Declaration

public interface CiscoObjectContainer

Fields

None

Methods

Table 139: Methods in CiscoObjectContainer

Interface	Method	Description
java.lang.Object	getObject()	Gets the application-defined object.
java.lang.Object	setObject(java.lang.Objectreference)	Sets an application-defined object.

Related Documentation

None

CiscoOutOfServiceEv

The CiscoOutOfServiceEv event is the super class for the out-of-service events CiscoAddrOutOfServiceEv and CiscoTermOutOfServiceEv. This class defines the causes for out-of-service events.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, javax.telephony.events.Ev

Subinterfaces

CiscoAddrOutOfServiceEv, CiscoTermOutOfServiceEv

Declaration

public interface CiscoOutOfServiceEv extends CiscoEv

Fields

Table 140: Fields in CiscoOutOfServiceEv

Interface	Field	Description
staticint	CAUSE_CALLMANAGER_FAILURE	The cause for this event is due a Cisco Unified Communications Manager failure.
staticint	CAUSE_CTIMANAGER_FAILURE	The cause for this event is due to a failure from CTIManager.
staticint	CAUSE_DEVICE_FAILURE	The cause for this event is a device failure.
staticint	CAUSE_DEVICE_RESTRICTED	The cause for this event is that the device is restricted.
staticint	CAUSE_DEVICE_UNREGISTERED	The cause for this event is that the device is in an unregistered state.

Interface	Field	Description
staticint	CAUSE_LINE_RESTRICTED	The cause for this event is that the line is restricted.
staticint	CAUSE_NOCALLMANAGER_AVAILABLE	The cause for this event is the unavailability of any Cisco Unified Communications Manager.
staticint	CAUSE_REHOME_TO_HIGHER_PRIORITY_CM	The cause for this event is an to error in failback to a higher-priority Cisco Unified Communications Manager node.
staticint	CAUSE_REHOMING_FAILURE	The cause for this event is a failure while attempting to rehome.
staticint	ID	—

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Methods

None

Related Documentation

See Constant Field Values, on page 1661

CiscoPartyInfo

This interface defines the party info of the call.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoPartyInfo

Fields

Table 141: Fields in CiscoPartyInfo

Interface	Field	Description
Static int	ABBREVIATED_NUMBER	This NumberType is same as 4; it represents caller is from same Cisco Unified Communications Manager server.
Static int	INTERNATIONAL_NUMBER	This NumberType is same as 0; it represents nothing is configured
Static int	NATIONAL_NUMBER	This NumberType is same as 1; it represents caller is INTERNATIONAL
Static int	NET_SPECIFIC_NUMBER	This NumberType is same as 2; it represents caller is NATIONAL
Static int	RESERVED_FOR_EXTENSION	This NumberType is same as 6; it represents its a fast dial call - not being used currently
Static int	SUBSCRIBER_NUMBER	This NumberType is same as 3; it represents call is from MGCP/H.323 gateway
Static int	UNKNOWN_NUMBER	—

Methods

Table 142: Methods in CiscoPartyInfo

Interface	Method	Description
javax.telephony.Address	getAddress()	Returns the address.
boolean	getAddressPI()	Returns Presentation Indicator (PI) associated with Address. If it returns true, Application can display this Address name to end users. If it returns false, Applications should not display this Address name to end user.
java.lang.String	getDisplayName()	Returns display name of the party.

Interface	Method	Description
boolean	getDisplayNamePI()	Returns the PI associated with DisplayName. If it returns true, Application can display this DisplayName to end users. If it returns false, Applications should not display this DisplayName to end user.
int	getlocale()	Returns the locale of the party unicode display name.
int	getNumberType()	Returns number type of the party.
java.lang.String	getUnicodeDisplayName()	Returns unicode display name of the party.
CiscoUrlInfo	getUrlInfo()	Returns URL Info.
java.lang.String	getVoiceMailbox()	Returns voice mail box of the party.

Related Documentation

See Constant Field Values, on page 1661.

CiscoPickupGroup

CiscoPickupGroup is a new interface that represents a Pickup Group at the JTAPI layer. Currently, all a PickupGroup is a pair of String objects representing the Pickup Group's DN, and the Pickup Group's Partition.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.0(1)	Following APIs are added: • getPickupGroupDN() • getPickupGroupPartition()

Declaration

public interface CiscoPickupGroup

Methods

Table 143: Methods in CiscoPickup Group

Interface	Method	Description
String	getPickupGroupDN()	Returns a String object that represents the number of the Pickup Group.
String	getPickupGroupPartition()	Returns a String object that represents the partition of the Pickup Group. It returns an empty String object if this pickup group does not belong to a partition.

Related Documentation

None

CiscoProvCallParkEv

CiscoProvCallParkEv event is delivered to providerobserver when a call is parked/unparked from any device in the cluster. To receive this event application should register using CiscoProvider.registerFeature() and CiscoProvFeatureID.MONITOR_CALLPARK_DN. User profile used by the application should have the Call Park Retrieval Allowed flag enabled to receive this event.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, CiscoProvFeatureEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoProvCallParkEv extends CiscoProvFeatureEv

Fields

Table 144: Fields in CiscoProvCallParkEv

Interface	Field	Description
Static int	ID	—

Interface	Field	Description
Static int	PARK_STATE_ACTIVE	Indicates that a call is parked.
staticint	PARK_STATE_IDLE	Indicates that a call is unparked.
staticint	REASON_CALLPARK	Indicates that this event is due to call park.
staticint	REASON_CALLPARKREMAINDER	Deprecated This interface is deprecated due to a spelling error. Use the new interface REASON_CALLPARKREMINDER.
staticint	REASON_CALLPARKREMINDER	Indicates that the call is offered back to the parking party after call park reminder.
staticint	REASON_CALLUNPARK	Indicates that the call is unparked.

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 145: Methods in CiscoProvCallParkEv

Interface	Method	Description
int	getintCallIDValue()	Returns an integer representation of this object.
java.lang.String	getParkDN()	Returns where the call is parked.
java.lang.String	getParkedParty()	Returns the DN of the parked party.

Interface	Method	Description
java.lang.String	getParkedPartyPartition()	Returns the partition of the Parked Party.
java.lang.String	getParkingParty()	Returns the DN of the parking party.
java.lang.String	getParkingPartyPartition()	Returns the partition of the Parking party.
java.lang.String	getParkPartyPartition()	Returns the partition of park DN.
int	getReason()	Returns the reason of the event.
int	getState()	Returns the state of the call. Possible states are CiscoProvCallParkEv.PARK_STATE_IDLE CiscoProvCallParkEv.PARK_STATE_ACTIVE.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoProvFeatureEv

getFeatureID

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoProvEv

The CiscoProvEv interface, which extends JTAPI's core javax.telephony.events.ProvEv interface, serves as the base interface for all Cisco-extended JTAPI Provider events. Every Provider-related event in this package extends this interface, directly or indirectly.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	Added new API getCiscoCause() which returns the CiscoCause for delivering the provider events.

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Subinterfaces

CiscoAddrActivatedEv, CiscoAddrActivatedOnTerminalEv, CiscoAddrAddedToTerminalEv, CiscoAddrCreatedEv, CiscoAddrRemovedEv, CiscoAddrRemovedFromTerminalEv, CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv, CiscoProvCallParkEv, CiscoProvConnToLeastPriorCtiServerEv, CiscoProvFallbackToPrimNwCompltdEv, CiscoProvFeatureEv, CiscoProvPrimNwReachableEv, CiscoProvTerminalCapabilityChangedEv, CiscoRestrictedEv, CiscoTermActivatedEv, CiscoTermCreatedEv, CiscoTermRemovedEv, CiscoTermRestrictedEv,

Declaration

public interface CiscoProvEv extends CiscoEv, javax.telephony.events.ProvEv

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Interface	Method	Description
int	getCiscoCause ()	This method returns the cause to let application know why the event has been delivered.
Static final int	CiscoProvEv.CAUSE_NORMAL	This indicates the cause for non - EM login/logout scenarios. It will have an integer value of 0.
Static final int	CiscoProvEv.CAUSE_EM_LOGIN	This cause indicates an EM login on a terminal with a profile that is in the application's control list and/or with a user id that matches with the user id with which application has been started. It will have an integer value of 1.
Static final int	CiscoProvEv. CAUSE_EM_LOGOUT	This cause indicates an EM logout from a terminal with the profile that is in the application's control list and/or with a user id that matches with the user id with which application has been started. It will have an integer value of 2.
Static final int	CiscoProvEv. CAUSE_EM_LOGIN_PROFILE_ADD	This cause indicates a case where profile is added to the control list when it is already logged into a terminal. It will have an integer value of 3.
Static final int	CiscoProvEv. CAUSE_EM_LOGIN_PROFILE_REMOVE	This cause indicates a case where profile is removed from the control list when it is already logged into a terminal. It will have an integer value of 4.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

CiscoProvFeatureEv

The CiscoProvFeatureEv interface extends the com.cisco.jtapi.extensions.CiscoProvEv interface for provider events.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Subinterfaces

CiscoProvCallParkEv

Declaration

public interface CiscoProvFeatureEv extends CiscoProvEv

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 146: Methods in CiscoProvFeatureEv

Interface	Method	Description
int	getFeatureID()	The feature ID for which the application wants to receive events.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoProvEv.

ProvEv.

CiscoProvFeatureID

This interface lists the features that registerFeature supports.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
7.1(3)	Interface is enhanced to allow application register to get CiscoProvTerminalRegisteredEv and CiscoProvTerminalUnRegisteredEv events when terminal register and unregister respectively. CiscoProvTerminalRegisteredEv and CiscoProvTerminalUnRegisteredEv will be delivered to Provider observer when application registers for this feature

Declaration

public interface CiscoProvFeatureID

Fields

Table 147: Fields in CiscoProvFeatureID

Interface	Field	Description
static int	MONITOR_CALLPARK_DN	Used in the registerFeature interface in CiscoProvider to receive CiscoProvCallParkEv when a call gets parked or unparked from any device in the cluster.
public static final int	TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY	Application can use to this to receive CiscoProvTerminalRegisteredEv and CiscoProvTerminalUnRegisteredEv

Sample Code

To register for Terminal Register and Unregister event notification:

```
try{JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
} catch (JtapiPeerUnavailableException e){

}

MyProviderObserver providerObserver  =  new MyProviderObserver ();
Try{
  provider  =  peer.getProvider ( ipaddress;login = useid;passwd = password );
} catch (ProviderUnavailableException exp){
}
  if ( provider ! =  null ) {
    provider.addObserver ( providerObserver );
    provInService.waitTrue();
    System.out.Println("Enabling Register and Unregister events  ");
    try{
((CiscoProvider)provider).registerFeature(CiscoProvFeatureID.
TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY);
   } catch (InvalidStateException ec){
   }

}
```

// CiscoProvTerminalRegisteredEv and CiscoProvTerminalUnRegisteredEv are delivered to Provider Observer.

```
class MyProviderObserver implements ProviderObserver {

  public synchronized void providerChangedEvent ( ProvEv [] eventList ) {
```

```
    try {
      if ( eventList ! =  null ) {
        for ( int i  =  0; i < eventList.length; i++ ) {
          if ( eventList[i] instanceof CiscoProvTerminalRegisteredEv ){
           CiscoProvTerminalRegisteredEv  ev  =  (CiscoProvTerminalRegisteredEv)

              eventList[i];
            System.out.Println( ev.getTerminal().getName() + " registered with
CUCM" );
          }
        }
     } catch (Exception eee){
}
```

Methods

None

Related Documentation

See Constant Field Values, on page 1661.

CiscoProvPickupCallAlertEv

CiscoProvPickupCallAlertEvent is a new interface being added with Call Pickup feature development. This event is fired whenever there is a call to be picked up in a pickup group that the provider is observing. See previous changes to CiscoProvider for information about how to register to observe a pickup group.

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New interface.

Declaration

public interface CiscoProvPickupCallAlertEvent extends CiscoProvEv

Methods

Table 148: Methods of CiscoProvPickupCallAlertEv

Interface	Method	Description
String	getPickupGroupNumber()	This method returns the Pickup Group Number for which this event is being fired.
String	getPickupGroupPartition()	This method returns the Pickup Group Number for which this event is being fired.
CiscoCallID	getCallID()	This method returns the Call ID for the ringing call.

Interface	Method	Description
CiscoPartyInfo	getCallingPartyInfo()	This method returns a CiscoPartyInfo representing the calling party.CAVEAT: Currently, if the calling party is from out of cluster (External), it will still report as being Internal on the Address object inside of the CiscoPartyInfo.
CiscoPartyInfo	getCalledPartyInfo()	This method returns a CiscoPartyInfo representing the called party.

CiscoProvTerminalIPAddressChangedEv

This interface will be delivered to provider observers added by applications whenever the IP address of a terminal changes without the terminal getting unregistered.

Interface History

Cisco Unified Communications Manager Release Number	Description
9.0(1)	New interface.

Declaration

public interface CiscoProvTerminalIPAddressChangedEv extends CiscoProvEv

Fields

Table 149: Fields in CiscoProvTerminalIPAddressChangedEv

Interface	Field	Description
public Terminal	getTerminal()	Returns the Terminal that registered with Cisco Unified Communication Manager.
public int	getIPAddressingMode()	Returns the active IP Addressing mode of the terminal after the change. Based on this value, applications can query either the Ipv4 or the Ipv6 address of the terminal. Addressing mode may be any of the following constants: CiscoTerminal.IP_ADDRESSING_MODE_IPv4 CiscoTerminal.IP_ADDRESSING_MODE_IPv6 CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6

Interface	Field	Description
public InetAddress	getIPV4Address()	Returns the IPv4 address of the terminal. If the addressing mode is CiscoTerminal.IP_ADDRESSING_MODE_IPv6, this method will return null.
public InetAddress	getIPV6Address()	Returns the IPv6 address of the terminal. If the addressing mode is CiscoTerminal.IP_ADDRESSING_MODE_IPv4, this method will return null.

Methods

None

Related Documentation

None

CiscoProvTerminalMultiMediaCapabilityChangedEv

CiscoProvTerminalMultiMediaCapabilityChangedEv is a new interface that is notified to application as a Provider Event. when the video capability of the terminal changes, this interface is delivered to the provider observers added by applications.

Table 150: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoProvTerminalMultiMediaCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoProvTerminalMultiMediaCapabilityChangedEv

Methods

Table 151: Methods in CiscoProvTerminalMultiMediaCapabilityChangedEv

Interface	Method	Description
Terminal	getTerminal()	This API returns the Terminal that is registered with Cisco UCM.

Interface	Method	Description
int	getVideoCapability()	This returns the video capability of the Terminal. The video capability can be: • CiscoMultiMediaCapabilityInfo.NONE • CiscoMultiMediaCapabilityInfo.VIDEO_ENABLED

CiscoProvTerminalRegisteredEv

This event is delivered to provider observer whenever a terminal registers with Cisco Unified Communication Manager. To receive this event, the application must use registerFeature API with CiscoFeatureID. TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY. This event is delivered if a Terminal registers to Cisco Unified Communication Manager after the application registers for the feature using registerFeature API. During initialization time and JTAPI failover time the application can see this event for some the Terminals in the control list.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(3)	New interface.

Declaration

public interface CiscoProvTerminalRegisteredEv extends CiscoProvEv.

Fields

Table 152: Fields in CiscoProvTerminalRegisteredEv

Interface	Field	Description
Terminal	getTerminal()	Returns the terminal that registered with Cisco Unified Communications Manager.

Methods

None

Related Documentation

None

CiscoProvTerminalUnRegisteredEv

This event is delivered to provider observer when ever a terminal unregisters from Cisco Unified Communication Manager. To receive this event, the application must use the registerFeature API with CiscoFeatureID. TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(3)	New interface.

Declaration

public interface CiscoProvTerminalUnRegisteredEv extends CiscoProvEv.

Fields

Table 153: Fields in CiscoProvTerminalRegisteredEv

Interface	Field	Description
Terminal	getTerminal()	Returns the terminal that un-registered with Cisco Unified Communications Manager.
public final static int public final static int public final static int public final static int	• REASON_UNKNOWN • REASON_RESET • REASON_LOGIN • REASON_LOGOUT	• Indicates Terminal un-registered for unknown reason • Indicates Terminal un-registered due to rest • Indicates Terminal un-registered due to login • Indicates Terminal un-registered due to logout
Int	getReason()	Returns the reason of un-register. The return value is one of the above defined reasons.

Methods

None

Related Documentation

None

CiscoProvider

The CiscoProvider interface extends the Provider interface with additional Cisco capabilities.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.0(1)	Enhanced to have the following: • New API registerPickupAlert(String pickupDn, String pickupPartition) • unregisterPickupAlert(String pickupDn, String pickupPartition) which allow the application to register and unregister for the reception of Call Pickup events. • CiscoProvPickupCallAlertEvent, which is a provider event what the application receives when they register for events using the previously mentioned API • ProviderCallPickupRegistrationClosedEv, which is a provider event used to alert the application if something happens that would close the registration event, such as the pickup group being removed from the Unified CM admin panel.
10.0(1)	A new method is added: getClusterID() this returns the clusterID enterprise parameter configured for the cluster as a string.
14SU3	Enhanced to have the following: • New APIs setLeastPriorityCtiServer(String leastPriorityCtiServer) • setLeastPriorityCtiServer(String leastPriorityCtiServer, int fallbackInitiationTime) • getLeastPriorityCtiServer() • isCtiServerAvailable(String server) • initiateFallback() • initiateFallback(String server)

Superinterfaces

CiscoObjectContainer, javax.telephony.Provider.

Declaration

public interface CiscoProvider extends javax.telephony.Provider, CiscoObjectContainer

Fields

None

Inherited Fields

From Interface javax.telephony.Provider

IN_SERVICE, OUT_OF_SERVICE, SHUTDOWN

New Error Codes

CTIERR_ALREADY_REGISTERED

This error code indicates that the Pickup Group attempting to be registerred for has already been registerred by this provider.

CTIERR_REGISTRATION_NOT_FOUND

This error code indicates that an unregister attempt failed because the Pickup Group specified was not registerred for previously.

CTIERR_INVALID_PICKUPGROUP

This error code indicates that the Pickup Group specified in the register or unregister event is not valid.

Methods

Table 154: Methods in CiscoProvider

Interface	Method	Description
CiscoTerminal	createTerminal (java.lang.Stringname)	Returns an instance of the CiscoTerminal class which corresponds to the given name. Application must have sufficient capability otherwise PrivilegeViolationException gets thrown CiscoProvider.createTerminal(). **Pre-conditions** this.getState() = = Provider.IN_SERVICE **Post-conditions** Create CiscoTerminal corresponding to name; terminal is an element of this.getTerminals(). **Parameters** • name—The name of desired CiscoTerminal object. **Throws** javax.telephony.InvalidArgumentException—The name provided does not correspond to a name of any CiscoMediaTerminal known to the Provider or within the Provider's domain. javax.telephony.InvalidStateException—The provider is not inService. PreviledgeVoilationException—The provider does not have sufficient capbilitly i.e. CiscoProviderCapabilities.canObserveAnyTerminal() returns false call.getState() = = Call.INVALID

Interface	Method	Description
void	deleteTerminal (CiscoTerminalterminal)	Removes the CiscoTerminal Object from providers control. Application must have created this terminal using Provider.createTerminal() interface otherwise PreviledgeVoilationException gets thrown. CiscoProvider.deleteTerminal(). **Pre-conditions** this.getState() = = Provider.IN_SERVICE **Post-conditions** CiscoTerminal Object deleted from providers list of terminal. Terminal is not element of this.getTerminals() any more and Addresses belonging to terminal get deleted. **Parameters** • terminal—The referece to the desired CiscoTerminal object to be deleted. **Throws** javax.telephony.InvalidArgumentException—The terminal provided is not element of this.getTerminals() or terminal is not provider domain. PrivilegeViolationException—The terminal given in the argument is not a terminal created using Provider.createTerminal() method. Applications can delete only those terminal which are created using Provider.createTerminal() interface.
javax.telephony.Address	getAddress (java.lang.Stringnumber, java.lang.Stringpartition)	Returns an address object corresponding to the number and partition that is passed in the method. The address object will be unique for a particular number, partition combination. **Throws** javax.telephony.InvalidArgumentException
int	getAppDSCPValue ()	Gets the DSCP value from the provider by using CiscoProvider.getAppDSCPValue(). **Pre-conditions** this.getState() = = Provider.IN_SERVICE **Post-conditions** The method will return the integer value of the DSCP value for applications set by CTI.
CiscoCall	getCall (CiscoRTPHandlertpHandle)	Returns call object with the RTPHandle associated with a specific terminal.

Interface	Method	Description
CiscoCall	getCall (intcallleg)	Returns CiscoCall present in provider domain and the call object with the RTPHandle associated with a specific terminal. This method may return null if this RTPHandle is no longer associated with any call or if there was no callObserver added on the terminal at the time when CiscoCallOpenLogicalChannelEv which contained this handle is sent to applications. **Throws** javax.telephony.InvalidStateException
boolean	getCallbackGuardEnabled ()	None
boolean	isFIPSCompliantJTAPI ()	Returns true if JTAPI is running in FIPS Compliance mode. This means that the application has explicitly requested FIPS compliance, and that the libraries are running properly.
boolean	isFIPSCompliantCUCM ()	Returns true if the Unified CM server is running in FIPS Compliance mode.
CiscoIntercomAddress[]	getIntercomAddresses ()	Returns array of CiscoInterComAddress present in provider domain.

Interface	Method	Description
CiscoMediaTerminal	getMediaTerminal (java.lang.Stringname)	Returns an instance of the CiscoMediaTerminal class which corresponds to the given name. Each CiscoMediaTerminal has a unique name associated with it, which is assigned to it by the JTAPI implementation. If no CiscoMediaTerminal is available for the given name within the Provider domain, this method throws the InvalidArgumentException. This CiscoMediaTerminal is contained in the arrays generated by Provider.getTerminals() and CiscoProvider.getMediaTerminals(). **Pre-conditions** Let CiscoMediaTerminal terminal = this.getMediaTerminal(name); terminal is an element of this.getTerminals(); terminal is an element of this.getMediaTerminals(); **Post-conditions** Let CiscoMediaTerminal terminal = this.getMediaTerminal(name); terminal is an element of this.getTerminals(); terminal is an element of this.getMediaTerminals(); **Parameters** • name—The name of desired CiscoMediaTerminal object. **Throws** javax.telephony.InvalidArgumentException—The name provided does not correspond to a name of any CiscoMediaTerminal known to the Provider or within the Provider domain.

Interface	Method	Description
CiscoMediaTerminal[]	getMediaTerminals ()	Returns an array of CiscoMediaTerminals associated with the Provider and within the Provider local domain. Each CiscoMediaTerminal possesses a unique name, which is assigned to it by the JTAPI implementation. If there are no CiscoMediaTerminals associated with this Provider, then this method returns null. This array is a subset of the array returned by Provider.getTerminals(). **Post-conditions** Let CiscoMediaTerminal[] terminals = this.getMediaTerminals() terminals = = null or terminals.length > = 1 if terminals ! = null, terminals is a subset of this.getTerminals () **Throws** javax.telephony.ResourceUnavailableException—Indicates the number of media terminals present in the Provider is too great to return as a static array.
CiscoPickupGroup[]	getRegisteredPickupGroups ()	This method returns an array of CiscoPickupGroup objects that represents all of the Pickup Groups that this provider is currently registerred to observe. **Parameters** A String object that represents the number of the Pickup Group to be registerred for, and another String object that represents the partition that the Call Pickup Group is in.
java.lang.String	getVersion ()	None
void	registerFeature (intfeatureID)	Registers a particular feature for which application gets Provider events. Applications should pass in the featureID of the softkey. Current supported features are listed in CiscoProvFeatureID interface. **Throws** javax.telephony.InvalidStateException javax.telephony.PrivilegeViolationException javax.telephony.InvalidArgumentException

Interface	Method	Description
void	registerPickupAlert (String pickupDN, String pickupPartition)	This method tells the Provider to register for receiving Call Pickup events. After this method is called, Call Pickup events for the specified Call Pickup Group will be sent to all JTAPI observers under this provider. **Parameters** • A String object that represents the number of the Pickup Group to be registerred for, and another String object that represents the partition that the Call Pickup Group is in. • The pickupPartition can be passed in as an empty String (“”) or null if the pickup group does is not in any partition. • An application can use the new CiscoPickupGroup object in place of the pair of Strings for either method.
void	registerPickupAlert (CiscoPickupGroup pickupGroup)	This method tells the Provider to register for receiving Call Pickup events. After this is called, Call Pickup events for the specified Call Pickup Group will be sent to all JTAPI observers under this provider. **Parameters** • A String object that represents the number of the Pickup Group to be registerred for, and another String object that represents the partition that the Call Pickup Group is in. • The pickupPartition can be passed in as an empty String (“”) or null if the pickup group does is not in any partition. • An application can use the new CiscoPickupGroup object in place of the pair of Strings for either method.

Interface	Method	Description

Void	setCallbackGuardEnabled (booleanenabled)	Enables or disables try/catch logic for observer callbacks. In order to protect itself from application exceptions in observer callbacks, the Provider normally guards all invocations of application interfaces (e.g. observers) with the following code:
		`try {observer.callStateChanged ( ... );`
		`} catch ( Throwable t ) {`
		`// log the exception here`
		`}`
		This isolates application errors from the JTAPI implementation, allowing easier troubleshooting, since the JTAPI implementation can note the unhandled exception and continue operating.
		Some errors are considered non-recoverable and will be re-thrown by JTAPI, generally resulting in application exit. Such errors include ThreadDeath, OutOfMemoryError, and StackOverflowError.
		Applications wishing to trap errors within JTAPI threads should create a subclass of ThreadGroup and initialize JTAPI from a thread within that ThreadGroup.
		By overriding the ThreadGroup.uncaughtException () method, the application can be made aware of all unrecoverable errors thrown on JTAPI threads. In some cases, JTAPI's aggressive error-catching approach may make it more difficult to troubleshoot applications within a java debugger.
		Microsoft Visual J++ version 6.0, for example, does not handle breakpoints within application observer callbacks properly if JTAPI catches Throwable. In such cases, JTAPI application developers may choose to disable the internal JTAPI try/catch logic.
		Note Disabling callback guards in this manner is only intended for use while troubleshooting applications, and never for use in production environments. By default, callback guards are always enabled.
		Parameters
		• enabled—if true, callback guard will be enabled; if false, callback guard will be disabled.

Void	unregisterFeature (intfeatureID)	Unregisters a particular feature.

Interface	Method	Description
Void	unregisterPickupAlert (String pickupDN, String pickupPartition)	This method will tell the Provider to unregister for receiving Call Pickup events. After this is called, Call Pickup events for the specified Call Pickup Group will no longer be sent to all JTAPI observers under this provider. **Parameters** • A String object that represents the number of the Pickup Group to be registerred for, and another String object that represents the partition that the Call Pickup Group is in. • The pickupPartition can be passed in as an empty String (“”) or null if the pickup group does is not in any partition. • An application can use the new CiscoPickupGroup object in place of the pair of Strings for either method.
Void	unregisterPickupAlert (CiscoPickupGroup pickupGroup)	This method will tell the Provider to unregister for receiving Call Pickup events. After this is called, Call Pickup events for the specified Call Pickup Group will no longer be sent to all JTAPI observers under this provider. **Parameters** • A String object that represents the number of the Pickup Group to be registerred for, and another String object that represents the partition that the Call Pickup Group is in. • The pickupPartition can be passed in as an empty String (“”) or null if the pickup group does is not in any partition. • An application can use the new CiscoPickupGroup object in place of the pair of Strings for either method.
CiscoRemoteTerminal[]	getRemoteTerminals ()	This method returns an array of CiscoRemoteTerminal associated with the Provider and within the Provider's domain. Each CiscoRemoteTerminal possesses an unique name, which is assigned to it by the JTAPI implementation. If there is no CiscoRemoteTerminals associated with this Provider, this API will return null. This array is a subset of the array returned by Provider.getTerminals().

Interface	Method	Description
CiscoRemoteTerminal	getRemoteTerminal (String name)	This method returns an instance of the CiscoRemoteTerminal class which corresponds to the given name. Each CiscoRemoteTerminal has an unique name associated to it, which is assigned by the JTAPI implementation. If no CiscoRemoteTerminal is available for the given name within the Provider's domain, this API throws the InvalidArgumentException. This CiscoRemoteTerminal is contained in the arrays generated by Provider.getTerminals() and CiscoProvider.getRemoteTerminals().
String	getClusterID ()	This method returns the clusterID enterprise parameter configured for the cluster as a string. If this enterprise parameter is changed CTIManager service and other Cisco Communication Manager services need to be restarted. **Pre-conditions** Provider.getState() = = Provider.IN_SERVICE If pre-condition is not met, InvalidStateException is thrown. Default value is StandAloneCluster.
Void	setLeastPriorityCtiServer(String leastPriorityCtiServer)	This method allows application to mark a CTI server as least priority. Least Priority indicates, JTAPI would connect to the CTI server only in the case when no other CTI Server configured by application is reachable. JTAPI would also initiate a forceful fallback once one of the CTI servers are reachable again (600 seconds post this event). Basically, all other CTI servers configured and not marked as least priority have equal weightage. Application can only configure one CTI server as least priority.

Interface	Method	Description
void	setLeastPriorityCtiServer(String leastPriorityCtiServer, int fallbackInitiationTime)	This API allows application to mark a CTI server as least priority. Least Priority indicates, JTAPI would connect to the CTI server only in the case when no other CTI Server configured by application is reachable. JTAPI would also initiate a forceful fallback once one of the CTI servers arereachable again. Basically, all other CTI servers configured and not marked as least priority have equal weightage. Application can only configure one CTI server as least priority. This method is overloaded to take additional parameter to specify time (in seconds) after which a forceful fallback is to be initiated once primary network becomes reachable. Fallback initiation time is defined as below: • Default value : 300 seconds • Min value : 120 seconds • Max value: 600 seconds • Default value would be taken if specified value is out of the range.
String	getLeastPriorityCtiServer()	This API returns the least priority CTI server as configured by application by invoking <CODE>CiscoProvider.setLeastPriorityCtiServer((server)</CODE>.
boolean	isCtiServerAvailable(String server)	This API allows application to query if one of the configured CTI servers is reachable.
void	initiateFallback()	This API allows application to invoke a fallback when connected to CTI server which was previously identified as least priority by invoking <CODE>CiscoProvider.setLeastPriorityCtiServer(server)</CODE>. Application can invoke this in case one of the primary network CTI Server becomes available and application is ready to do a fallback before the configured/default fallback timer expires.
void	initiateFallback(String server)	This API allows application to invoke a fallback when connected to CTI server which was previously identifie as least priority by invoking <CODE>CiscoProvider.setLeastPriorityCtiServer(server)</CODE>. Application can invoke this in case one of the primary network CTI Server becomes available and application is ready to do a fallback before the configured/default fallback timer expires. This method is overloaded to take additional parameter to specify the server to which application needs to fallback to.

Inherited Methods

From Interface javax.telephony.Provider

addObserver, createCall, getAddress, getAddressCapabilities, getAddressCapabilities, getAddresses, getCallCapabilities, getCallCapabilities, getCalls, getCapabilities, getConnectionCapabilities, getConnectionCapabilities, getName, getObservers, getProviderCapabilities, getProviderCapabilities, getState, getTerminal, getTerminalCapabilities, getTerminalCapabilities, getTerminalConnectionCapabilities, getTerminalConnectionCapabilities, getTerminals, removeObserver, shutdown

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

None

CiscoProviderCapabilities

This interface defines the Cisco-specific provider capabilities that Cisco Unified JTAPI offers.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added support for the method canSupportIPv6()T.
8.0(1)	Enhanced by adding new API canAutoPickup(), which lets the application determine whether or not the CUCM service parameter "Auto Call Pickup Enabled" is set to true or false. This service parameter has an impact on the events and behavior of Call Pickup, and applications can use this new API to determine if it's enabled or not, and act accordingly.

Superinterfaces

javax.telephony.capabilities.ProviderCapabilities

Declaration

public interface CiscoProviderCapabilities extends javax.telephony.capabilities.ProviderCapabilities

Methods

Table 155: Methods in CiscoProviderCapabilities

Interface	Method	Description
boolean	canAutoPickup()	This method returns a boolean value representing whether or CUCM service parameter "Auto Call Pickup Enabled" is set ot true or false.
boolean	canObserveAnyTerminal()	This method checks whether the user has been provisioned in the Cisco Unified Communications Manager with the privilege to observe any Terminal (and its addresses) in the system. Such Terminals and Addresses do not get returned as part of the list that JTAPI initializes at startup. The provider obtained with the login for a user with such privileges can be determined from the canObserverAnyTerminal method call in ProviderCapabilities. Returns True if the user can observe any Terminal in the system, or false if the user can only observe Terminals and Addresses in the control list.

	Example
	`Provider p = peer.getProvider( loginString );`
	`ProviderCapabilities caps = p.getCapabilities ();`
	`if ( caps instanceof CiscoProviderCapabilities )`
	`{`
	`boolean canObserveAnyTerminal =`
	`((CiscoProviderCapabilities)caps).canObserveAnyTerminal ();`
	`boolean canMonitorParkDN =`
	`((CiscoProviderCapabilities)caps).canMonitorParkDNs ();`
	`boolean canModifyCallingPN =`
	`((CiscoProviderCapabilities)caps).canModifyCallingParty ();`
	`boolean canRecordCalls =`
	`((CiscoProviderCapabilities)caps).canRecord();`
	`boolean canMonitorCalls =`
	`((CiscoProviderCapabilities)caps).canMonitor();`
	`}`	

boolean	canMonitorParkDNs()	This method checks whether the user has been provisioned in the Cisco Unified Communications Manager to monitor park DNs. Returns True if the user can monitor park DNs, or false otherwise.
boolean	canModifyCallingParty()	This method checks whether the user has been provisioned in the Cisco Unified Communications Manager to modify the calling party number of a call. Returns True if the user can modify the calling party number, or false otherwise.

Interface	Method	Description
boolean	canRecord()	This method checks whether the user has been provisioned in the Cisco Unified Communications Manager to record calls. Only users in 'Standard CTI Allow Call Recording' user group can record calls. Returns True if the user belongs to the group.
boolean	canMonitor()	This method checks whether a user has been provisioned in the Cisco Unified Communications Manager to monitor calls. Only users in 'Standard CTI Allow Call Monitoring' user group can initiate call monitoring request. Returns True if the user belongs to the group.
boolean	canSupportIPv6()	This interface returns true if Enterprise Parameter “Enable IPv6” is enabled and false otherwise.

Inherited Methods

From Interface javax.telephony.capabilities.ProviderCapabilities

isObservable

Related Documentation

See canObserveAnyTerminal().

CiscoProviderCapabilityChangedEv

Application provider observers receive this event when a user gets added or removed from user groups (capabilitied) in Cisco Unified Communications Manager. The methods for this event let you check which capabilities changed.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added hasIPv6CapabilityChanged() method.

Declaration

public interface CiscoProviderCapabilityChangedEv

Fields

Table 156: Fields in CiscoProviderCapabilityChangedEv

Interface	Field	Description
staticint	ID	None
staticint	MODIFY_CGPN	Deprecated This constant is not returned by any interface, should not be used by application.
staticint	MONITOR_PARKDN	Deprecated This constant is not returned by any interface, should not be used by application.
staticint	SUPERPROVIDER	Deprecated This constant is not returned by any interface, should not be used by application.

Methods

Table 157: Methods in CiscoProviderCapabilityChangedEv

Interface	Method	Description
CiscoProviderCapabilities	getCapability()	This method returns the current CiscoProviderCapabilities object for the user.
boolean	hasIPv6CapabilityChanged()	This method can be used by applications to determine whether Enable IPv6 Enterprise Parameter has changed. **Pre-conditions** this.getState() = = Provider.IN_SERVICE **Post-conditions** The method returns True when the Enable IPv6 Enterprise parameter gets changed; otherwise it returns False.
boolean	hasModifyCallingPartyChanged()	This method checks whether the “modify Calling Party" privilege has changed. **Pre-conditions** provider.getState() = = Provider.IN_SERVICE

Interface	Method	Description
boolean	hasMonitorCapabilityChanged()	This method checks whether the monitor capability of a user has changed. **Pre-conditions** provider.getState() = = Provider.IN_SERVICE
boolean	hasMonitorParkDNChanged()	This method checks whether the "monitor Park DN" privilege has changed. **Pre-conditions** provider.getState() = = Provider.IN_SERVICE
boolean	hasObserveAnyTerminalChanged()	This method checks whether the "can control any terminal" privilege has changed **Pre-conditions** provider.getState() = = Provider.IN_SERVICE
boolean	hasRecordingCapabilityChanged()	This method checks whether the recording capability of the has changed. **Pre-conditions** provider.getState() = = Provider.IN_SERVICE

Related Documentation

See Constant Field Values, on page 1661.

CiscoProviderObserver

Implement this interface to receive CiscoProvEv events such as CiscoAddrCreatedEv and CiscoTermCreatedEv when observing a Provider via the Provider.addObserver method.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.ProviderObserver

Declaration

public interface CiscoProviderObserver extends javax.telephony.ProviderObserver

Methods

None

Inherited Methods

From Interface javax.telephony.ProviderObserver

providerChangedEvent

Related Documentation

See CiscoAddrCreatedEv and CiscoTermCreatedEv.

CiscoProvTerminalCapabilityChangedEv

This event is delivered to the Provider when Terminal Capability is changed. This event is provided on application observer .

Interface History

Cisco Unified Communications Manager Release Number	Description
7.0(1)	Added event.
7.0(1)	Modified the CiscoTerminal[] interface so that only CiscoMediaTerminals or CiscoRouteTerminals gets returned.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoProvTerminalCapabilityChangedEv extends CiscoProvEv

Fields

Table 158: Fields in CiscoProvTerminalCapabilityChangedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 159: Methods in CiscoProvTerminalCapabilityChangedEv

Interface	Method	Description
CiscoTerminal[]	getTerminals()	Returns an array of CiscoTerminals whose capabilities have changed. In Cisco Unified Communications Manager Release 7.0(1), CiscoTerminal[] interface was modified so that only CiscoMediaTerminals or CiscoRouteTerminals get returned.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

None

CiscoProvTerminalRemoteDestinationChangedEv

CiscoProvTerminalRemoteDestinationChangedEv is a new interface exposed to the applications as a Provider Event. JTAPI sends this event to the application's provider observer when any of Remote Destination information is changed on a CiscoRemoteTerminal in the provider domain. This event contains the latest list of Remote Destinations at a given time. And depending on the request or operation done on these remote destinations, single or multiple CiscoProvTerminalRemoteDestinationChangedEv event(s) can be generated from the change notification(s).

Methods

Interface	Method	Description
CiscoRemoteTerminal	getTerminal()	This method returns the CiscoRemoteTerminal object for which its remote destination has changed.
CiscoRemoteDestinationInfo[]	getRemoteDestinations()	This method returns an array of CiscoRemoteDestinationInfo objects representing the latest/current list of remote destinations associated to the CiscoRemoteTerminal at the time when this change notification event took place.
boolean	isMyAppLastToSetActiveRD()	This method can be used by application to determine whether it is the last application to set active remote destination for the CiscoRemoteTerminal.

CiscoRecorderInfo

This interface provides information about the recorder in a recording session. When a recording session is active, this interface gives information about the recording device.

Interface History

Cisco Unified Communications Manager Release Number	Description
12.5(1)	A new method getMultiForkingRecorderInfo() is added which returns an array (maximum size 5) of CiscoMultiForkingRecorderInfo[]. The following are the four new methods inside getMultiForkingRecorderInfo() that provide the information about each recorder. • getRecorderURI() • getRecorderErrorMsg() • getRecorderType() • getRecorderStatus()
10.0(1)	Four new methods are added: • getMediaForkingDeviceType() • getMediaForkingDeviceName() • getProtocolReferenceGUID() • getMediaForkingClusterID()
9.0(1)	A new method, getRecordingType(), is added.
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRecorderInfo

Fields

None

Methods

Table 160: Methods in CiscoRecorderInfo

Interface	Method	Description
CiscoAddress	getAddress()	Returns the recorder address.
public int	getRecordingType()	This method returns the recording type that was used to start the recording.
java.lang.String	getTerminalName()	Returns the terminal name of the recording device.

Interface	Method	Description
public int	getMediaForkingDeviceType()	This interface returns the Media Forking Device Type for Gateway Recording. The forking device type can be: CiscoCall.CALL_RECORDING_MEDIA_FORKING _DEVICE_TYPE_NONE = 0 CiscoCall.CALL_RECORDING_MEDIA_FORKING _DEVICE_TYPE_PHONE = 1 CiscoCall.CALL_RECORDING_MEDIA_FORKING _DEVICE_TYPE_GW = 2
java.lang.String	getMediaForkingDeviceName()	This interface returns the Forking Device Name for Gateway Recording.
java.lang.String	getProtocolReferenceGUID()	This interface returns the Protocol Call Reference GUID for Gateway Recording.
java.lang.String	getMediaForkingClusterID()	This interface returns the Forking Cluster ID for Gateway Recording.

Range of Values

The range of values returned by the getRecordingType() method is defined on the CiscoCall object:

CiscoCall.CALL_RECORDING_TYPE_NONE

CiscoCall.CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT

CiscoCall.CALL_RECORDING_TYPE_AUTOMATIC

CiscoCall.CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION

CiscoCall.CALL_RECORDING_TYPE_USER_INITIATED_FROM

Related Documentation

None

CiscoRemoteDestinationInfo

CiscoRemoteDestinationInfo is a new interface that contains the information about a remote destination on a CiscoRemoteTerminal. Applications can get a list of all associated remote destinations from the return array of CiscoRemoteTerminal.getAllRemoteDestinations().

Methods

Interface	Method	Description
String	getRemoteDestinationName()	This method returns the remote destination name.
String	getRemoteDestinationNumber()	This method returns the remote destination number.

Interface	Method	Description
boolean	getIsActiveRD()	This method returns whether the remote destination is an active remote destination or not.

CiscoRemoteTerminal

CiscoRemoteTerminal is a new interface that extends the interface of CiscoTerminal. A CiscoRemoteTerminal is a special kind of CiscoTerminal representing the CTI Remote Device and Jabber/CUCSF (Cisco Unified Client Services Framework) Device in extend mode. It allows applications to monitor remote destinated devices such as PSTN, PBSX, and Mobiles phones.

Interface History

Cisco Unified Communications Manager Release Number	Description
9.0(1)	A new interface, CiscoRemoteTerminal, is added.

Declaration

public interface CiscoRemoteTerminal

Methods

Interface	Method	Description
CiscoRemoteDestinationInfo[]	getAllRemoteDestinations ()	This method will return an array of CiscoRemoteDestinationInfo representing all remote destinations of the CiscoRemoteTerminal, or null if none. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown.
CiscoRemoteDestinationInfo[]	getActiveRemoteDestinations ()	This method will return an array CiscoRemoteDestinationInfo representing all active remote destinations of the CiscoRemoteTerminal, or null if none. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown.
void	setActiveRemoteDestination (String remoteDestinationNumber, boolean isActiveRD)	This method will set/unset an active remote destination of the CiscoRemoteTerminal based on the remote destination number. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. Also note that the Remote Destination Number must be of a valid associated remote destination, and if the remoteDestinationNumber parameter is null, it will throw InvalidArgumentException.

Interface	Method	Description
void	addRemoteDestination (String remoteDestinationName, String remoteDestinationNumber, boolean isActiveRD)	This method will add a new remote destination to the CiscoRemoteTerminal. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. And if either the remoteDestinationNumber or remoteDestinationName parameter is null, it will throw InvalidArgumentException.
void	removeRemoteDestination (String remoteDestinationNumber)	This method will remove a remote destination from the CiscoRemoteTerminal based on the remote destination number. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. Also note that the Remote Destination Number must be of a valid associated remote destination, and if the remoteDestinationNumber parameter is null, it will throw InvalidArgumentException.
void	removeAllRemoteDestinations ()	This API will remove all associated remote destinations from the CiscoRemoteTerminal. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown.
void	updateRemoteDestinationName (String remoteDestinationNumber, String remoteDestinationName)	This method will update the name of a remote destination of the CiscoRemoteTerminal based on the remote destination number. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. Also note that the Remote Destination Number must be of a valid associated remote destination, and if the remoteDestinationNumber parameter is null, it will throw InvalidArgumentException.
void	updateRemoteDestinationNumber (String remoteDestinationNumber, StringnewRemoteDestinationNumber)	This method will update the number of a remote destination of the CiscoRemoteTerminal based on the remote destination number. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. Also note that the Remote Destination Number must be of a valid associated remote destination, and if either the remoteDestinationNumber or newRemoteDestinationNumber parameter is null, it will throw InvalidArgumentException.

Interface	Method	Description
void	updateRemoteDestination (String remoteDestinationNumber, String remoteDestinationName, String newRemoteDestinationNumber, boolean isActiveRD)	This method will update a remote destination of the CiscoRemoteTerminal based on the remote destination number. It can update any or all of its remote destination name, remote destination number, and isActiveRD at the same time. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown. Also note that the Remote Destination Number must be of a valid associated remote destination, and if the remoteDestinationNumber parameter is null, it will throw InvalidArgumentException.
boolean	isRegisteredByThisApp ()	This method will return true if this application issued a successful registration request to register this terminal's device in Extend mode; return false otherwise. It will get set to true until this application unregisters the device.
int	getRegistrationType ()	This method will return the registration type with which this terminal's device has been registered in. The registration type returned can be one of the following: • CiscoRemoteTerminal. EXTEND_MEDIA_REGISTRATION • CiscoRemoteTerminal. NO_EXTEND_MEDIA_REGISTRATION
boolean	isMyAppLastToSetActiveRD ()	This method will return true if this application is the last application to set active remote destination for the CiscoRemoteTerminal; return false otherwise. Note that CiscoProvider must be in IN_SERVICE state, otherwise InvalidStateException will be thrown.

Parameters

String remoteDestinationName

The name of the specified remote destination.

String remoteDestinationNumber

The number of the specified remote destination.

boolean isActiveRD

The flag to indicate whether the specified remote destination is an active remote destination or not.

Data Type

public static final int

EXTEND_MEDIA_REGISTRATION = 8

This registration type applies to CUCSF device that is registered in Extend mode, which as a result is representing as a CiscoRemoteTerminal.

NO_EXTEND_MEDIA_REGISTRATION = 0

This registration type applies to non-CUCSF device that is static registered and is representing as CiscoRemoteTerminal, such as CTI Remote Device.

New Error Codes

CiscoJtapiException.CTIERR_INVALID_REMOTE_DESTINATION_NUMBER (0x8CCC0121)

CiscoJtapiException.CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER (0x8CCC0122)

CiscoJtapiException.CTIERR_REMOTEDESTINATION_LIMIT_EXCEEDED (0x8CCC0123)

CiscoJtapiException. CTIERR_REMOTE_DEVICE_REQUEST_FAILED_ACTIVE_RD_NOT_SET (0x 8CCC0124)

CiscoJtapiException.CTIERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE (0x 8CCC0126)

CiscoJtapiException.CTIERR_DEVICE_ALREADY_REGISTERED_NONEXTEND (0x 8CCC0127)

CiscoJtapiException.CTIERR_MEDIA_ALREADY_TERMINATED_EXTEND (0x 8CCC0128)

CiscoJtapiException.CTIERR_INVALID_REMOTE_DESTINATION_NAME (0x8CCC0130)

Sample Code

```
Sample Code:
public static void main(String[] args) {
  try
  {
    JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer(null);
    String provStr  =  provName + ";login = " + login + ";passwd = " + passwd;
    Provider myProvider  =  peer.getProvider(provStr);
    MyProviderObserver providerObserver  =  new MyProviderObserver();
    String myDevName  =  "CTIRD-A";
    String temp  =  “”;
    if (myProvider ! =  null)
    {
      myProvider.addObserver(providerObserver);
      provInService.waitTrue();
      CiscoRemoteTerminal rTerm  =
          (CiscoRemoteTerminal)(myProvider.getTerminal(myDevName));
      if (rTerm ! =  null)
      {
        CiscoRemoteDestinationInfo[] remoteDestinations  =
            rTerm.getAllRemoteDestinations();
        CiscoRemoteDestinationInfo[] activeRemoteDestinations  =
            rTerm.getActiveRemoteDestinations();
```

```
        System.out.println("CTI Remote Device Name: " + rTerm.getName());
        if (remoteDestinations ! =  null && remoteDestinations.length ! =  0)
        {
          System.out.println("Number of associated Remote Destinations (RD): " +
            remoteDestinations.length);
          for (int i  =  0; i < remoteDestinations.length; i++)
          {
            System.out.println("RD["+i+"] Name: " +
              remoteDestinations[i].getRemoteDestinationName());
            System.out.println("RD["+i+"] Number: " +
              remoteDestinations[i].getRemoteDestinationNumber());
            System.out.println("RD["+i+"] IsActiveRD: " +
              remoteDestinations[i].getIsActiveRD());
            temp  =  remoteDestinations[i].getRemoteDestinationName();
            if (temp ! =  null && temp.equalsIgnoreCase("MyOffice"))
            {
              temp  =  remoteDestinations[i].getRemoteDestinationNumber();
              rTerm.updateRemoteDestinationNumber(temp,  "9498231202");
              rTerm.setActiveRemoteDestination("9498231202",  true);
            }
          }
          rTerm.addRemoteDestination("MyHome",  "6267978244",  false);
        }
        else
        {
          System.out.println("There is no associated Remote Destinations (RD)");
        }
      }
      else
      {
        System.out.println("There is no CTI Remote Device of " + myDevName +
          " in this provider");
      }
    }
    else
    {
      System.out.println("Cannot create provider");
    }
  }
  catch (Exception e)
  {
    System.out.println("Exception caught for getting CTI Remote Device RD info!
" + e);
    if (e instanceof PlatformException)
    {
      switch (((CiscoJtapiException) e).getErrorCode())
      {
      case CiscoJtapiException.CTIERR_INVALID_REMOTE_DESTINATION_NUMBER:
        System.out.println("Invalid RD Number");
        break;
      case CiscoJtapiException. CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER:
        System.out.println("Duplicated RD Number");
        break;
      }
    }
  }
}
...
```

CiscoRestrictedEv

The CiscoRestrictedEv event is the parent class for the CiscoAddrRestrictedEv and CiscoAddrRestrictedOnTerminalEv events. This is the base class for restricted events and defines the cause codes for all restricted events.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Subinterfaces

CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv

Declaration

public interface CiscoRestrictedEv extends CiscoProvEv

Fields

Table 161: Fields in CiscoRestrictedEv

Interface	Field	Description
staticint	CAUSE_UNKNOWN	The Terminal is restricted for an unknown reason.
staticint	CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION	The Terminal is restricted due to an unsupported configuration (for example, configuring the rollover option).
staticint	CAUSE_UNSUPPORTED_PROTOCOL	The Terminal in the control list is using a protocol that Cisco Unified JTAPI does not support.
staticint	CAUSE_USER_RESTRICTED	The Terminal or Address is marked as restricted.

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

None

CiscoRouteAddress

This interface is deprecated and has not been implemented.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.Address, javax.telephony.callcenter.RouteAddress

Declaration

public interface CiscoRouteAddress extends javax.telephony.callcenter.RouteAddress

Fields

None

Inherited Fields

From Interface javax.telephony.callcenter.RouteAddress

ALL_ROUTE_ADDRESS

Methods

Table 162: Methods in CiscoRouteAddress

Interface	Method	Description
void	registerRouteCallback (javax.telephony.callcenter.RouteCallbackrouteCallback, booleandisableAutoRehoming)	Deprecated **Throws** javax.telephony.ResourceUnavailableException javax, telephony.MethodNotSupportedException

Inherited Methods

From Interface javax.telephony.callcenter.RouteAddress

cancelRouteCallback, getActiveRouteSessions, getRouteCallback, registerRouteCallback

From Interface javax.telephony.Address

addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

Related Documentation

None

CiscoRouteEvent

The CiscoRouteEvent interface extends the RouteEvent interface with additional Cisco-specific capabilities. Applications can use the getCallingPartyIpAddr method to obtain the IP address of the calling party device.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added the method getCallingPartyIpAddr_v6().

Superinterfaces

javax.telephony.callcenter.events.RouteEvent, javax.telephony.callcenter.events.RouteSessionEvent

Declaration

public interface CiscoRouteEvent extends javax.telephony.callcenter.events.RouteEvent

Fields

None

Inherited Fields

From Interface javax.telephony.callcenter.events.RouteEvent

SELECT_ACD, SELECT_EMERGENCY, SELECT_LEAST_COST, SELECT_NORMAL, SELECT_USER_DEFINED

Methods

Table 163: Methods in CiscoRouteEvent

Interface	Method	Description
java.net.InetAddress	getCallingPartyIpAddr_v6()	Returns the IPv6 address of the calling party. If the IP address is not available, this method returns an InetAddress with the IP address 0::0 and a null host name. Printing this object yields a string representation of "null/0::0". Returns: InetAddress.
java.net.InetAddress	getCallingPartyIpAddr()	Returns the IP address of the calling party. If the IP address is not available, this method returns an InetAddress with the IP address 0.0.0.0 and a null host name. Printing this object yields a string representation of "null/0.0.0.0".

Inherited Methods

From Interface javax.telephony.callcenter.events.RouteEvent

getCallingAddress, getCallingTerminal, getCurrentRouteAddress, getRouteSelectAlgorithm, getSetupInformation

From Interface javax.telephony.callcenter.events.RouteSessionEvent

getRouteSession

Related Documentation

None

CiscoRouteSession

The CiscoRouteSession interface supports application access to the underlying call that is associated with a RouteSession. Also, this interface exposes various internal ERRORs for RouteEndEvent.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
11.5(1)	Added route selection with deviceName

Superinterfaces

javax.telephony.callcenter.RouteSession

Declaration

public interface CiscoRouteSession extends javax.telephony.callcenter.RouteSession

Fields

Table 164: Fields in CiscoRouteSession

Interface	Field	Description
static final int	ERROR_ROUTESELECT_TIMEOUT	Each routeEvent() or reRouteEvent() that is sent starts a timer for the application to respond with a routeSelect() or endRoute(). The default value of this timer is 5 seconds. Should the application not respond within this time, the system calls an endRoute with this error.
static final int	ERROR_NO_CALLBACK	Because there is no default route mechanism in place, if there is no callback registered for this application, the system calls an endRoute with this error.
static final int	ERROR_INVALID_STATE	If an internal InvalidStateException occurred, or some preconditions or postconditions were not met during routing, the system calls endRoute with this error.
static final int	DEFAULT_SEARCH_SPACE	This indicates that the redirect should be done by using the search space that is the default for the implementation. The default is to use the caller search space.
static final int	CALLINGADDRESS_SEARCH_SPACE	This indicates that the redirect should be done by using the search space of the calling address.

Interface	Field	Description
static final int	ROUTEADDRESS_SEARCH_SPACE	This indicates that the redirect should be done by using the search space of the route point address.
static final int	DONOT_RESET_ORIGINALCALLED	This is a parameter value for the PreferedOriginalCalled option; it specifies not to reset OriginalCalled.
static final int	RESET_ORIGINALCALLED	This is a parameter value for PreferedOriginalCalled Option; if the value of preferedOriginalCalledOption is set to this, it will reset the OriginalCalled to preferedOriginalCalledNumber.
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_NEEDED	This constant returned by RouteSession.getCause() indicates that the routeSelectedElement in the selectRoute does not contain the required FAC code.
static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_NEEDED	This constant returned by RouteSession.getCause() indicates that the routeSelectedElement in the selectRoute does not contain the required CMC code.
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	This constant returned by RouteSession.getCause() indicates that the routeSelectedElement in the selectRoute does not contain the required FAC and CMC codes.
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_INVALID	This constant returned by RouteSession.getCause() indicates that the routeSelectedElement in the selectRoute contains an invalid FAC code.

Interface	Field	Description
static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_INVALID	This constant returned by RouteSession.getCause() indicates that the routeSelectedElement in the selectRoute contains an invalid CMC code.

Inherited Fields

From Interface javax.telephony.callcenter.RouteSession

CAUSE_INVALID_DESTINATION, CAUSE_NO_ERROR, CAUSE_PARAMETER_NOT_SUPPORTED, CAUSE_ROUTING_TIMER_EXPIRED, CAUSE_STATE_INCOMPATIBLE, CAUSE_UNSPECIFIED_ERROR, ERROR_RESOURCE_BUSY, ERROR_RESOURCE_OUT_OF_SERVICE, ERROR_UNKNOWN, RE_ROUTE, ROUTE, ROUTE_CALLBACK_ENDED, ROUTE_END, ROUTE_USED

Methods

Table 165: Methods in CiscoRouteSession

Interface	Method	Description
javax.telephony.Call	getCall()	Returns the call associated with this RouteSession.
void	selectRoute(java.lang. String[]routeSelected, intcallingSearchSpace)	Overloads the selectRoute method in the RouteSession interface to allow applications to specify a calling search space to use when the call is redirected to the route destination. **Parameters** javax.telephony. MethodNotSupportedException **Throws** • callingSearchSpace—One of CiscoRouteSession. DEFAULT_SEARCH_SPACE; CiscoRouteSession. CALLINGADDRESS_SEARCH_SPACE; or CiscoRouteSession. ROUTEADDRESS_SEARCH_SPACE. • routeSelected—A list of possible destinations for the call.

Interface	Method	Description
void	selectRoute(java.lang. String[]routeSelected, intcallingSearchSpace, java.lang. String[]modifyingCallingNumber)	

Interface	Method	Description
		Selects one or more possible routing destinations for a call with a modified calling number. This method takes as an argument that is a string array of destination telephone address names, modifyingCallingNumber, arranged in priority order. The highest-priority destination is the first element in the specified array and routing is attempted with this destination first with the corresponding element of modifying calling number. If modifiedCallingNumber is null for an element, the calling number is not modified if a call is routed to that particular routeSelected element. The system attempts to use the specified destination addresses in order until the system successfully selects a destination. The system delivers a RouteUsedEvent to the application when it routes the call to that destination. **Pre-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = RouteSession.ROUTE or RouteSession.RE_ROUTE **Post-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = • RouteSession.ROUTE_USED (if the Call was successfully routed.) A RouteUsedEvent gets delivered for this RouteSession if a successful destination was selected. **Parameters** • routeSelected—Possible destinations for callcallingSearchSpace can be CiscoRouteSession.DEFAULT_SEARCH_SPACE; CiscoRouteSession. CALLINGADDRESS_SEARCH_SPACE; or CiscoRouteSession. ROUTEADDRESS_SEARCH_SPACE. • modifyingCallingNumber—An array of elements for which the application wants to modify the calling number when the call reaches the routeSelected element.

Interface	Method	Description
		Throws • com.cisco.jtapi. MethodNotSupportedExceptionImpl (The implementation does not support routing.) • javax.telephony. PrivilegeViolationException (The user does not have the necessary privileges to use this method.) • javax.telephony.MethodNotSupportedException selectRoute

Interface	Method	Description
void	selectRoute(java.lang. String[]routeSelected, intcallingSearchSpace, java.lang. String[]preferedOriginalCalledNumber, int[]preferedOriginalCalledOption	

Interface	Method	Description
		Selects one or more possible destinations for routing the Call. This method takes as an argument a string array of destination telephone address names, in prioritized order, and a string array for the PreferredOriginalCalled number. PreferedOriginalCalled number gets selected based on the index of the destination telephone names array. If the index corresponding to the destination array is not found in the PreferedOriginalCalled number array, preferedOriginalCalled gets set to the destination. The highest-priority destination is the first element in the given array, and the system attempts to route with this destination first. The system attempts each given destination address in succession until the call gets successfully routed. The system delivers a RouteUsedEvent event to the application when a successful routing destination has been selected and the Call has been routed to that destination. **Pre-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = RouteSession.ROUTE or • RouteSession.RE_ROUTE **Post-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = • RouteSession.ROUTE_USED (if the Call was successfully routed.) A RouteUsedEvent gets delivered for this RouteSession if a successful destination was selected. **Parameters** • routeSelected—Possible destinations for the call. • preferedOriginalCalledNumber—List with each item corresponding to a route at the matching array index in the routeSelected list. • preferedOriginalCalledOption—List of options, each corresponding to routeSelected list. The option specifies whether to set OriginalCalled to preferedOriginalCalledNumber. The option values

Interface	Method	Description
		are CiscoRouteSession. DONOT_RESET_ORIGINALCALLED and CiscoRouteSession. RESET_ORIGINALCALLED. If the value is unspecified or null, the default is CiscoRouteSession. DONOT_RESET_ORIGINALCALLED. **Throws** com.cisco.jtapi. MethodNotSupportedExceptionImpl (The implementation does not support routing.) javax.telephony. PrivilegeViolationException (The user does not have the necessary privileges to use this method.) javax.telephony. MethodNotSupportedException selectRoute

Interface	Method	Description
void	selectRoute(java.lang. String[]routeSelected, intcallingSearchSpace, java.lang. String[]modifyingCallingNumber, java.lang. String[]preferedOriginalCalledNumber, int[]preferedOriginalCalledOption, java.lang. String[]facCode, java.lang. String[]cmcCode)	

Interface	Method	Description
		Selects one or more possible routing destinations. It takes as arguments a string array of: • destination telephone address names, in prioritized order • PreferredOriginalCalled numbers • FACs • CMCs The system selects the PreferedOriginalCalled number corresponding to the index of the destination telephone name array. If the index is not found in the PreferedOriginalCalled number array, the preferedOriginalCalled gets set to the destination. The highest priority destination is the first element in the specified array, and the system attempts to route the call to that destination first. The system attempts the specified destination addresses in order, until the call gets routed successfully. The system delivers a RouteUsedEvent event to the application when the system has selected a successful routing destination and routed the call to that destination. **Pre-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = RouteSession.ROUTE • RouteSession.RE_ROUTE **Post-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = • RouteSession.ROUTE_USED (if the call was successfully routed). A RouteUsedEvent gets delivered for this RouteSession if a successful destination was selected. **Parameters** • routeSelected—List of possible destinations. • preferedOriginalCalledNumber—List with each member of corresponding to the route at the same array index in the routeSelected. • list.preferedOriginalCalledOption—List of options,

Interface	Method	Description
		each corresponding to RouteList. The option specifies whether to set OriginalCalled to preferedOriginalCalledNumber. The option values are CiscoRouteSession. DONOT_RESET_ORIGINALCALLED and CiscoRouteSession. RESET_ORIGINALCALLED.If the value is unspecified or null, the default is CiscoRouteSession. DONOT_RESET_ORIGINALCALLED. • modifyingCallingNumber—Array of elements for which the application wants modify the calling number when the call reaches the routeSelected element. If applications do not want to modify the number, a null value for this parameter must be passed by the application. • facCode (Forced Authorization Code [FAC])—If a routeSelected element requires a FAC, the corresponding facCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter. • cmcCode - (Client Matter Code [CMC]) If a routeSelected element requires a CMC, the corresponding cmcCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter. **Throws** • com.cisco.jtapi. MethodNotSupportedExceptionImpl (The implementation does not support routing.) • javax.telephony. PrivilegeViolationException (The user does not have the necessary privileges to use this method.) • javax.telephony. MethodNotSupportedException selectRoute

Interface	Method	Description
void	selectRoute(java.lang. String[]routeSelected, intcallingSearchSpace, java.lang. String[]modifyingCallingNumber, java.lang. String[]preferedOriginalCalledNumber, int[]preferedOriginalCalledOption, java.lang. String[]facCode, java.lang. String[]cmcCode, intfeaturePriority)	

Interface	Method	Description
		Selects one or more possible routing destinations. It takes a string array of: • Destination telephone address names, in prioritized order • PreferredOriginalCalled numbers • FACs • CMCs • Integer priorities The PreferedOriginalCalled number gets selected based on the index of the destination telephone name array. If the index tis not found in the PreferedOriginalCalled number array, preferedOriginalCalled gets set to the destination. The highest-priority destination is the first element in the given array, and the system attempts to route with this destination first. The system tries the successive specified destination addresses until the call gets routed successfully. The system delivers a RouteUsedEvent event to the application when a successful routing destination has been selected and the call has been routed to that destination. **Pre-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = RouteSession.ROUTE • RouteSession.RE_ROUTE **Post-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = • RouteSession.ROUTE_USED (if the Call was successfully routed.) **Parameters** routeSelected—Possible destinations for the call. preferedOriginalCalledNumber—List with each element corresponding to the route at the same array index in the routeSelected list. preferedOriginalCalledOption—Options list, each corresponding to RouteList. The option specifies whether

Interface	Method	Description

		to set OriginalCalled to preferedOriginalCalledNumber. The option values are CiscoRouteSession. DONOT_RESET_ORIGINALCALLED and CiscoRouteSession. RESET_ORIGINALCALLED. If the value is unspecified or null, the default is CiscoRouteSession. DONOT_RESET_ORIGINALCALLED.
		• modifyingCallingNumber—Array of elements for which the application would like to modify the calling number when the call reaches the routeselected element. If applications do not want to modify the number, they must pass a null value for this parameter.
		• facCode (Forced Authorization Code [FAC])—If a routeSelected element requires a FAC, the corresponding facCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter.
		• cmcCode (Client Matter Code [CMC])—If a routeSelected element requires a CMC, the corresponding cmcCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter.
		• featurePriority—Sets the feature priority of the call. The application may set CiscoCall. FEATUREPRIORITY_NORMAL if the application does not want to set any specific priority. The featurePriority parameter may be:
		  • CiscoCall. FEATUREPRIORITY_NORMAL
		  • CiscoCall. FEATUREPRIORITY_URGENT
		  • CiscoCall. FEATUREPRIORITY_EMERGENCY
		Throws
		• javax.telephony. PrivilegeViolationException (The user does not have the necessary privileges to use this method.)
		• com.cisco.jtapi. MethodNotSupportedExceptionImpl (The implementation does not support routing.)
		• javax.telephony. MethodNotSupportedException

Interface	Method	Description
void	selectRoute(java.lang. String[]routeSelected, int[]callingSearchSpace, java.lang. String[]modifyingCallingNumber, java.lang. String[]preferedOriginalCalledNumber, int[]preferedOriginalCalledOption, java.lang. String[]facCode, java.lang. String[]cmcCode, int[]featurePriority)	

Interface	Method	Description
		Selects one or more possible routing destinations. It takes a string array of: • destination telephone address names, in prioritized order • calling search spaces • modifyingCallingNumbers • PreferredOriginalCalled numbers • FACs • CMCs • feature priorities The PreferedOriginalCalled number gets selected based on the index of the destination telephone name array. If the index tis not found in the PreferedOriginalCalled number array, preferedOriginalCalled gets set to the destination. The highest-priority destination is the first element in the given array, and the system attempts to route with this destination first. The system tries the successive specified destination addresses until the call gets routed successfully. The system delivers a RouteUsedEvent event to the application when a successful routing destination has been selected and the call has been routed to that destination. **Pre-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = RouteSession.ROUTE • RouteSession.RE_ROUTE **Post-conditions** • this.getRouteAddress().getProvider().getState() = = • Provider.IN_SERVICE this.getState() = = • RouteSession.ROUTE_USED (if the call was successfully routed.) **Parameters** • routeSelected—List of possible destinations for the call. • callingSearchSpace—For each route selected; can

Interface	Method	Description
		be CiscoRouteSession. DEFAULT_SEARCH_SPACE, CiscoRouteSession. CALLINGADDRESS_SEARCH_SPACE, or CiscoRouteSession. ROUTEADDRESS_SEARCH_SPACE. • preferedOriginalCalledNumber—List with each element corresponding to the route at the same array index in the routeSelected list. • preferedOriginalCalledOption—Options list, each corresponding to RouteList. The option specifies whether to set OriginalCalled to preferedOriginalCalledNumber. The option values are CiscoRouteSession. DONOT_RESET_ORIGINALCALLED and CiscoRouteSession. RESET_ORIGINALCALLED. If the value is unspecified or null, the default is CiscoRouteSession. DONOT_RESET_ORIGINALCALLED. • modifyingCallingNumber—Elements array for which the application would like to modify the calling number when the call reaches the routeselected element. If applications do not want to modify the number, they must pass a null value for this parameter. • facCode (Forced Authorization Code [FAC])—If a routeSelected element requires a FAC, the corresponding facCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter. • cmcCode (Client Matter Code [CMC])—If a routeSelected element requires a CMC, the corresponding cmcCode element must contain that code. If no code is required for a routeSelected element, the application must pass a null value for this parameter. • featurePriority—For each route selected, the feature priority can be set. The application may set CiscoCall. FEATUREPRIORITY_NORMAL if the application does not want to set any specific priority. The featurePriority parameter may be CiscoCall. FEATUREPRIORITY_NORMAL, CiscoCall. FEATUREPRIORITY_URGENT, or CiscoCall.FEATUREPRIORITY_EMERGENCY **Throws**

Interface	Method	Description
		• javax.telephony. PrivilegeViolationException (The user does not have the necessary privileges to use this method.) • com.cisco.jtapi. MethodNotSupportedExceptionImpl (The implementation does not support routing.) • javax.telephony. MethodNotSupportedException
void	selectRoute(String[] routeSelected, int[] callingSearchSpace, String[] modifyingCallingNumber,String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption, String[] facCode, String[] cmcCode,int[] featurePriority, byte[][] applicationXMLData, String[] deviceName	This method is similar to the above method, but takes an array of destination device names, in a prioritized order. This order of device names corresponds to the order of destinations provided in route selected. This method takes a string array of: • destination telephone address names, in prioritized order • calling search spaces • modifyingCallingNumbers • PreferredOriginalCalled numbers • FACs • CMCs • feature priorities • Application XML • device Names

Inherited Methods

From Interface javax.telephony.callcenter.RouteSession

endRoute, getCause, getRouteAddress, getState, selectRoute

Related Documentation

See Constant Field Values, on page 1661.

CiscoRouteTerminal

A CiscoRouteTerminal is a special kind of CiscoTerminal that allows applications to terminate RTP media streams. Unlike a CiscoTerminal, a CiscoRouteTerminal does not represent a physical telephony endpoint,

which is observable and controllable in a third-party manner. Instead, a CiscoRouteTerminal is a logical telephony endpoint that may be associated with any application that wants to route calls and terminate media. Unlike a CiscoMediaTerminal, a CiscoRouteTerminal can have multiple active calls at the same time. Typically, applications use CiscoRouteTerminals to queue calls until an agent is available to service them.

Note CiscoRouteTerminals are CTI Route Points on Cisco Unified Communications Manager.

Terminating media is a three-step process as follows:

1. The application registers its media capabilities with this Terminal by using the CiscoRouteTerminal.register method.
2. The application adds an observer that implements the CiscoTerminalObserver interface by using the Terminal.addObserver method.
3. The application must call addCallObserver on the CiscoRouteTerminal or the CiscoRouteAddress to receive and answer calls.

Applications will receive a CiscoMediaOpenLogicalChannelEv for each call or whenever media is stopped and needs to be reestablished. Applications must supply an IP address and port number by using the setRTPParams method on CiscoRouteTerminal.

Note **Important**—All applications written for or prior to CiscoJtapiClient Release 1.4 must be modified to register with CiscoRouteTerminal.NO_MEDIA_TERMINATION type if the applications are not interested in media termination.

Multiple applications can register with same RoutePoint as long as they are registered with the same media capabilities and registration type. All applications, if registered with CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION, will receive CiscoMediaOpenLogicalChannelEv when they add a callObserver, but only one application will be able to invoke setRTPParams.

Applications that are interested in media termination must add a CallObserver on the RouteAddress or on the CiscoRouteTerminals. Applications must not register with Dynamic type and add a registerRouteCallBack. Applications should only use registerRouteCallBack if they are not interested in media termination. Applications must not add a registerRouteCallBack and a callObserver at the same time.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Added these modes to the activeAddressingMode: • CiscoTerminal.IP_ADDRESSING_MODE_IPv6 • CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6

Superinterfaces

CiscoObjectContainer, CiscoTerminal, javax.telephony.Terminal

Declaration

public interface CiscoRouteTerminal extends CiscoTerminal

Fields

Table 166: Fields in CiscoRouteTerminal

Interface	Field	Description
staticint	DYNAMIC_MEDIA_REGISTRATION	Applications that are interested in media termination need to register with this type and pass in the capabilities that the application supports in the registration request.
staticint	NO_MEDIA_REGISTRATION	Applications that are not interested in media termination need to register with this type and pass in a null value for CiscoMediaCapability in the registration request. If registrationType is CiscoRouteTerminal.NO_MEDIA_REGISTRATION, the application cannot terminate media and can use the CiscoRouteTerminal for call routing.

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoTerminal

ASCII_ENCODING, DEVICESTATE_ACTIVE, DEVICESTATE_ALERTING, DEVICESTATE_HELD, DEVICESTATE_IDLE, DEVICESTATE_UNKNOWN, DEVICESTATE_WHISPER, DND_OPTION_CALL_REJECT, DND_OPTION_NONE, DND_OPTION_RINGER_OFF, IN_SERVICE, IP_ADDRESSING_MODE_IPV4, IP_ADDRESSING_MODE_IPV4_V6, IP_ADDRESSING_MODE_IPV6, IP_ADDRESSING_MODE_UNKNOWN, IP_ADDRESSING_MODE_UNKNOWN_ANATRED, NOT_APPLICABLE, OUT_OF_SERVICE, UCS2UNICODE_ENCODING, UNKNOWN_ENCODING

Methods

Table 167: Methods in CiscoRouteTerminal

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities, intregistrationType)	Registers a Terminal with specified CiscoMediaCapabilities and register type. The Provider must be in the Provider.IN_SERVICE state. Returns successfully when the CiscoRouteTerminal is registered. **Parameters** • capabilities—List of RTP encodings that the application supports for this Terminal. If the application is not interested in media termination, you may pass in a null value. • registrationType—Either CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION or CiscoRouteTerminal .NO_MEDIA_REGISTRATION. If registrationType is CiscoRouteTerminal .NO_MEDIA_REGISTRATION, the application cannot terminate media and can use the CiscoRouteTerminal for call routing. If registrationType is CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION, the application can terminate media and can have multiple active calls. This registrationType indicates that the application will supply the IP address and port dynamically for each call. Applications registering with this type receive a CiscoMediaOpenLogicalChannelEv for each call and must supply the IP address and port number by using the setRTPParams method on the CiscoRouteTerminal . **Throws** • CiscoRegistrationException

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities, intregistrationType, int[]algorithmIDs)	Registers a Terminal with the specified CiscoMediaCapabilities, registrationType, and supported SRTP algorithms. The Provider must be in the Provider.IN_SERVICE state. This method returns successfully when the CiscoRouteTerminal is registered. **Parameters** • capabilities—List of RTP encodings that the application supports for this Terminal. If the application is not interested in media termination, you may pass in a null value. • registrationType—Either CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION or CiscoRouteTerminal .NO_MEDIA_REGISTRATION. If registrationType is CiscoRouteTerminal .NO_MEDIA_REGISTRATION, the application cannot terminate media and can use the CiscoRouteTerminal for call routing. Other parameters in the method are ignored. If registrationType is CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION, the application can terminate media and can have multiple active calls. This registrationType indicates that the application will supply the IP address and port dynamically for each call. Applications registering with this type receive a CiscoMediaOpenLogicalChannelEv for each call and must supply the IP address and port number by using the setRTPParams method. • algorithmIDs—List of SRTP algorithms that the application supports for this Terminal. To use this, the application must have the TLS Link and SRTP Enabled flag enabled. AlgorithmIDs must be one of CiscoMediaEncryptionSupportedAlgorithms. **Throws** • javax.telephony.PrivilegeViolationException (The application tried to use the method, but is not authorized to use it.) • CiscoRegistrationException

Interface	Method	Description
void	register(CiscoMediaCapability[]capabilities, intregistrationType, int[]algorithmIDs, intactiveAddressingMode)	

Interface	Method	Description
		The CiscoRouteTerminal must be in the CiscoTerminal.UNREGISTERED state and its Provider must be in the Provider.IN_SERVICE state. The successful effect of this method is to register the RouteTerminal. Registers a Terminal with specified CiscoMediaCapabilities and register type and supported SRTP Algorithms. If registrationType is CiscoRouteTerminal .NO_MEDIA_REGISTRATION, application cannot terminate media and can use route point for call routing purpose. Other parameters in the method are ignored. If registration Type is CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION, then app can terminate media and can have multiple active calls. This Indicates that application is interested in supplying ipAddress and port dynamically for each call. Applications registering with this type will receive CiscoMediaOpenLogicalChannelEv for each call and will have to supply ipAddress and port number using setRTPParams method on CiscoRouteTerminal . **Method arguments** Capabilities indicates the type of RTP encodings that the application is willing to support for this Terminal. If application is not intersted in media termination, it may pass in null value registrationType may be CiscoRouteTerminal .NO_MEDIA_REGISTRATION or CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION. Supported Algorithms may be the SRTP Algorithms that application supports for this terminal. In order to use this, application need to have TLS Link and SRTP Enabled flag enabled. PrivilegeViolationException is thrown if app is not authorized to use this method. **Post-condition** This method returns successfully when the CiscoRouteTerminal is registered. **Parameters** • capabilities—List of RTP encodings supported by this terminal. • registrationType—CiscoRouteTerminal .DYNAMIC_MEDIA_REGISTRATION or CiscoRouteTerminal .NO_MEDIA_REGISTRATION

Interface	Method	Description

		• algorithmIDs—List of supported SRTP algorithms. AlgorithmIDs may only be one of CiscoMediaEncryptionSupportedAlgorithms.
		• activeAddressingMode—IP Addressing mode in which application intends to register this CiscoRouteTerminal . The modes can be:
		  • CiscoTerminal.IP_ ADDRESSING_MODE_IPv4
		  • CiscoTerminal.IP_ ADDRESSING_MODE_IPv6
		  • CiscoTerminal.IP_ ADDRESSING_MODE_IPv4_v6
		Throws
		• CiscoRegistrationException
		• javax.telephony.PrivilegeViolationException

void	unregister()	The CiscoRouteTerminal must be registered and its Provider must be in the Provider.IN_SERVICE state. The successful effect of this method is to unregister the CiscoRouteTerminal . **Post-condition** • This method returns successfully when the MediaTerminal is unregistered. **Throws** • CiscoUnregistrationException

Interface	Method	Description
void	setRTPParams(CiscoRTPHandlertpHandle, CiscoRTPParamsrtpParams)	Applications can set the IP address and RTP Port number to dynamically stream media for a call. To do this, applications must register MediaTerminal or CiscoRouteTeminal by providing only capabilities. Applications must then invoke this method upon receiving CiscoCallOpenLogicalChannelEv on the TerminalObserver. **Parameters** rtpHandle—The handle the application receives in CiscoCallOpenLogicalChannelEvrtpParams. Refer to CiscoRTPParams. **Throws** • javax.telephony.InvalidStateException • javax.telephony.InvalidArgumentException • javax.telephony.PrivilegeViolationException
boolean	isRegistered()	This method returns true if the CiscoMediaTerminal is registered and false otherwise. If the CiscoRouteTerminal is OutOfService, this method returns false; if it is InService, this method returns true. For CTIManager failure cases, this method returns false.
boolean	isRegisteredByThisApp()	This method returns true if this application issued a successful registration request. The registration remains valid even if the device is out-of-service because of a CTIManager failure. This returns true until this application unregisters the device.
int	getIPAddressingMode()	Application can invoke this API to query the IP Addressing Mode of the CiscoRouteTerminal . Addressing mode may be any of the following constants: • CiscoTerminal.IP_ ADDRESSING_IPv4 • CiscoTerminal.IP_ ADDRESSING_IPv6 • CiscoTerminal.IP_ ADDRESSING_IPv4_v6

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoTerminal

createSnapshot, getAltScript, getDeviceState, getDNDOption, getDNDStatus, getEMLoginUsername, getFilter, getLocale, getProtocol, getRegistrationState, getRTPInputProperties, getRTPOutputProperties, getState, getSupportedEncoding, isRestricted, sendData, sendData, setDNDStatus, setFilter, unPark

From Interface javax.telephony.Terminal

addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

getObject, setObject

Related Documentation

See CiscoTerminal and Constant Field Values, on page 1661 CiscoMediaOpenLogicalChannelEv

CiscoRouteUsedEvent

The CiscoRouteUsedEvent event indicates that the RouteSession moved into the RouteSession.ROUTE_USED state and the call terminated at a destination as a result of application routing. This interface extends the RouteUsedEvent interface and gets reported via the RouteCallback interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.callcenter.events.RouteSessionEvent, javax.telephony.callcenter.events.RouteUsedEvent

Declaration

public interface CiscoRouteUsedEvent extends javax.telephony.callcenter.events.RouteUsedEvent

Fields

None

Methods

Table 168: Methods in CiscoRouteUsedEvent

Interface	Method	Description
Int	getRouteSelectedIndex()	Returns an array index of the route where the call got routed.

Inherited Methods

From Interface javax.telephony.callcenter.events.RouteUsedEvent

getCallingAddress, getCallingTerminal, getDomain, getRouteUsed

From Interface javax.telephony.callcenter.events.RouteSessionEvent

getRouteSession

Related Documentation

See RouteSession, RouteCallback, and RouteSessionEvent.

CiscoRTPBitRate

The RTPBitRate interface contains constants describing G.723 RTP bit rates. CiscoRTPInputProperties.getBitRate and CiscoRTPOutputProperties.getBitRate return these constants.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPBitRate

Fields

Table 169: Fields in CiscoRTPBitRate

Interface	Fields	Description
staticint	R5_3	This constant is the 5.3k G.723 bit rate.
staticint	R6_4	This constant is the 6.4k G.723 bit rate.

Methods

None

Related Documentation

See CiscoRTPInputProperties.getBitRate(), CiscoRTPOutputProperties.getBitRate()

CiscoRTPHandle

Use the CiscoRTPHandle object to get a call reference with CiscoProvider.getCall(CiscoRTPHandle). This object gets returned in CiscoMediaCallOpenLogicalChannelEv. Pass this handle in the setRTPParams parameter of CiscoMediaTerminal or CiscoRouteTerminal, depending on where the CiscoMediaCallOpenLogicalChannelEv event gets received.

If no call observer was added, or there was no call observer added at the time CiscoMediaCallOpen LogicalChannelEv got sent, CiscoProvider.getCall(CiscoRTPHandle) may return null.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPHandle

Fields

None

Methods

Table 170: Methods in CiscoRTPHandle

Interface	Method	Description
Int	getHandle()	Returns the Cisco Unified Communications Manager CallLeg ID of the call, in integer format.

Related Documentation

None

CiscoRTPInputKeyEv

The CiscoRTPInputKeyEv event interface gives the key information for the encrypted incoming media stream. Applications should set the filter by using CiscoTermEvFilter.setRTPKeyEventsEnabled(true) to get this event via the TerminalObserver.terminalChangedEvent().

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPInputKeyEv extends CiscoTermEv

Fields

Table 171: Fields in CiscoRTPInputKeyEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN,

META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 172: Methods in CiscoRTPInputKeyEv

Interface	Method	Description
int	getCiscoMediaEncryptionKeyInfo()	Returns CiscoMediaEncryptionKeyInfo only if the provider is opened with the TLS link and SRTP enabled options set for the application in the Cisco Unified Communications Manager administration. Otherwise, it returns null.
int	getCiscoMediaSecurityIndicator()	Returns the media security indicator, one of the following constants: CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_AVAILABLE CiscoMediaSecurityIndicator. MEDIA_ENCRYPT_USER_NOT_AUTHORIZED CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_UNAVAILABLE
CiscoCallID	getCallID()	Returns a CiscoCallID object if there is already a CiscoCall present when this event is sent. If there is no CiscoCall present, this method returns null. getCallID().getCall() gives the call for which this key applies.
int	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event got delivered, CiscoProvider.getCall returns null. Returns:CiscoRTPHandle.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoRTPParams, CiscoMediaSecurityIndicator.

CiscoRTPInputProperties

The CiscoRTPInputProperties interface returns the properties of the media received by the Terminal (the inbound media stream). CiscoRTPInputStartedEv indicates that the media started.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPInputProperties

Fields

None

Methods

Table 173: Methods in CiscoRTPInputProperties

Interface	Method	Description
int	getBitRate()	Returns the media bit rate and can CiscoRTPBitRate.R5_3 or CiscoRTPBitRate.R6_4.
boolean	getEchoCancellation()	Returns True if the application needs to use echo cancellation.
java.net.InetAddress	getLocalAddress()	Returns the address to which media will be directed.
int	getLocalPort()	Returns the port to which media will be directed.
int	getPacketSize()	Returns the packet size, in milliseconds.

Interface	Method	Description
int	getPayloadType()	Returns the payload format, which is one of the following constants: • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.WIDEBAND_256K

Related Documentation

See CiscoRTPPayload and CiscoRTPBitRate.

CiscoRTPInputStartedEv

The CiscoRTPInputStartedEv event indicates the start of incoming media.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPInputStartedEv extends CiscoTermEv

Fields

Table 174: Fields in CiscoRTPInputStartedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 175: Methods in CiscoRTPInputStartedEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns CiscoCallID.
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object.
int	getMediaConnectionMode()	Returns a CiscoMediaConnectionMode.
int	getRTPInputProperties()	Returns CiscoRTPInputProperties, which gives the characteristics of the incoming media.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoRTPInputProperties, CiscoCallID, CiscoRTPParams, and CiscoMediaConnectionMode.

CiscoRTPInputStoppedEv

The CiscoRTPInputStoppedEv event indicates that the incoming media stream has stopped.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPInputStoppedEv extends CiscoTermEv

Fields

Table 176: Fields in CiscoRTPInputStoppedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 177: Methods in CiscoRTPInputStoppedEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns CiscoCallID. CiscoRTPInputStartedEv applies to CiscoCallID.getCall().
CiscoRTPHandle	getCiscoRTPHandle()	Returns CiscoRTPHandle object. Applications can get call reference using CiscoProvider.getCall If there is no callobserver or there was no callobserver when this event is delivered, then CiscoProvider.getCall may return null.
int	getMediaConnectionMode()	Returns a CiscoMediaConnectionMode with one of the following values for mediaMode: • CiscoMediaConnectionMode. RECEIVE_ONLY (one-way media, receive only) • CiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE: (two-way media) In general, you should never get an event with mode NONE; however, if that happens, applications should ignore the event and log an error.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoMediaConnectionMode, CiscoCallID, and CiscoRTPParams.

CiscoRTPOutputKeyEv

The CiscoRTPOutputKeyEv event gives the key information for the encrypted outgoing (transmitted) media stream. Applications set the filter by using CiscoTermEvFilter.setRTPKeyEventsEnabled(true) to get this event via the TerminalObserver.terminalChangedEvent().

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputKeyEv extends CiscoTermEv

Fields

Table 178: Fields in CiscoRTPOutputKeyEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 179: Methods in CisctoRTPOutputKeyEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns a CiscoCallID object if there is already a CiscoCall present when this event is sent. If there is no CiscoCall present, this method returns null.
CiscoMediaEncryptionKeyInfo	getCiscoMediaEncryptionKeyInfo()	Returns CiscoMediaEncryptionKeyInfo only if the provider is opened with the TLS link and SRTP enabled options set for the application in Cisco Unified Communications Manager administration. Otherwise, it will return null.
int	getCiscoMediaSecurityIndicator()	Returns media security indicator, one of the following constants: • CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_AVAILABLE • CiscoMediaSecurityIndicator. MEDIA_ENCRYPT_USER_NOT_AUTHORIZED • CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_UNAVAILABLE

Interface	Method	Description
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoRTPParams, and CiscoMediaSecurityIndicator.

CiscoRTPOutputProperties

The CiscoRTPOutputProperties interface gives the properties of the media transmitted by the terminal. CiscoRTPOutputStartedEv indicates that the media has started.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPOutputProperties

Fields

None

Methods

Table 180: Methods in CiscoRTPOutputProperties

Interface	Method	Description
int	getBitRate()	Returns the media bit rate, one of the following constants: • CiscoRTPBitRate.R5_3 • CiscoRTPBitRate.R6_4
int	getMaxFramesPerPacket()	Returns the maximum number of frames to send per packet.
int	getPacketSize()	Returns the packet size, in milliseconds.
int	getPayloadType()	Returns the payload format, which is one of the following constants: • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE • CiscoRTPPayload.WIDEBAND_256K
int	getPrecedenceValue()	Returns the precedence value.
java.net. InetAddress	getRemoteAddress()	Returns the address to which media is to be transmitted.
int	getRemotePort()	Returns the port to which media is to be transmitted.
boolean	getSilenceSuppression()	Returns false if Cisco Unified Communication Manager service parameter “Silence Suppression” is set to False or True otherwise.

Related Documentation

See CiscoRTPBitRate.

CiscoRTPOutputStartedEv

The CiscoRTPOutputStartedEv event interface indicates the start of media transmission.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputStartedEv extends CiscoTermEv

Fields

Table 181: Fields in CiscoRTPOutputStartedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION,

CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 182: Methods in CiscoRTPOutputStartedEv

Interface	Method	Description
CiscoRTPOutputProperties	getRTPOutputProperties()	Returns the RTP output properties.
CiscoCallID	getCallID()	Returns CiscoCallID. CiscoRTPOutputStartedEv applies to CiscoCallID.getCall().
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.
int	getMediaConnectionMode()	Returns a CiscoMediaConnectionMode with one of the following values for mediaMode: • CiscoMediaConnectionMode. TRANSMIT_ONLY (one-way media; transmit only) • CiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE (two-way media) **Note** In general, you should never get an event with mode NONE; however, if that happens, applications should ignore the event and log an error.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoCallID, and CiscoRTPParams.

CiscoRTPOutputStoppedEv

The CiscoRTPOutputStoppedEv event indicates that the media transmission stopped.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputStoppedEv extends CiscoTermEv

Fields

Table 183: Fields in CiscoRTPOutputStoppedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION,

CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 184: Methods in CiscoRTPOutputStoppedEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns CiscoCallID. CiscoRTPOutputStoppedEv applies to CiscoCallID.getCall().
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.
int	getMediaConnectionMode()	• Returns CiscoMediaConnectionMode with one of the following values: • CiscoMediaConnectionMode. TRANSMIT_ONLY (one-way media; transmit) • onlyCiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE (two-way media) **Note** In general, you should never get an event with mode NONE; however, if that happens, applications should ignore the event and log an error.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoCallID, CiscoRTPParams, and CiscoMediaConnectionMode.

CiscoRTPOutputKeyEv

The CiscoRTPOutputKeyEv event gives the key information for the encrypted outgoing (transmitted) media stream. Applications set the filter by using CiscoTermEvFilter.setRTPKeyEventsEnabled(true) to get this event via the TerminalObserver.terminalChangedEvent().

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputKeyEv extends CiscoTermEv

Fields

Table 185: Fields in CiscoRTPOutputKeyEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN,

META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 186: Methods in CisctoRTPOutputKeyEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns a CiscoCallID object if there is already a CiscoCall present when this event is sent. If there is no CiscoCall present, this method returns null.
CiscoMediaEncryptionKeyInfo	getCiscoMediaEncryptionKeyInfo()	Returns CiscoMediaEncryptionKeyInfo only if the provider is opened with the TLS link and SRTP enabled options set for the application in Cisco Unified Communications Manager administration. Otherwise, it will return null.
int	getCiscoMediaSecurityIndicator()	Returns media security indicator, one of the following constants: • CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_AVAILABLE • CiscoMediaSecurityIndicator. MEDIA_ENCRYPT_USER_NOT_AUTHORIZED • CiscoMediaSecurityIndicator. MEDIA_ENCRYPTED_KEYS_UNAVAILABLE
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoRTPParams, and CiscoMediaSecurityIndicator.

CiscoRTPOutputProperties

The CiscoRTPPutputProperties interface gives the properties of the media transmitted by the terminal. CiscoRTPOutPutStartedEv indicates that the media has started.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPOutputProperties

Fields

None

Methods

Table 187: Methods in CiscoRTPOutputProperties

Interface	Method	Description
int	getBitRate()	Returns the media bit rate, one of the following constants: • CiscoRTPBitRate.R5_3 • CiscoRTPBitRate.R6_4
int	getMaxFramesPerPacket()	Returns the maximum number of frames to send per packet.
int	getPacketSize()	Returns the packet size, in milliseconds.

Interface	Method	Description
int	getPayloadType()	Returns the payload format, which is one of the following constants: • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE • CiscoRTPPayload.WIDEBAND_256K
int	getPrecedenceValue()	Returns the precedence value.
java.net.InetAddress	getRemoteAddress()	Returns the address to which media is to be transmitted.
int	getRemotePort()	Returns the port to which media is to be transmitted.
boolean	getSilenceSuppression()	Returns false if Cisco Unified Communication Manager service parameter “Silence Suppression” is set to False or True otherwise.

Related Documentation

See CiscoRTPBitRate.

CiscoRTPOutputStartedEv

The CiscoRTPOutputStartedEv event interface indicates the start of media transmission.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputStartedEv extends CiscoTermEv

Fields

Table 188: Fields in CiscoRTPOutputStartedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 189: Methods in CiscoRTPOutputStartedEv

Interface	Method	Description
CiscoRTPOutputProperties	getRTPOutputProperties()	Returns the RTP output properties.
CiscoCallID	getCallID()	Returns CiscoCallID. CiscoRTPOutputStartedEv applies to CiscoCallID.getCall().
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.
int	getMediaConnectionMode()	Returns a CiscoMediaConnectionMode with one of the following values for mediaMode: • CiscoMediaConnectionMode.TRANSMIT_ONLY (one-way media; transmit only) • CiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE (two-way media) **Note** In general, you should never get an event with mode NONE; however, if that happens, applications should ignore the event and log an error.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoCallID, and CiscoRTPParams.

CiscoRTPOutputStoppedEv

The CiscoRTPOutputStoppedEv event indicates that the media transmission stopped.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoRTPOutputStoppedEv extends CiscoTermEv

Fields

Table 190: Fields in CiscoRTPOutputStoppedEv

Interface	Field	Description
staticint	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 191: Methods in CiscoRTPOutputStoppedEv

Interface	Method	Description
CiscoCallID	getCallID()	Returns CiscoCallID. CiscoRTPOutputStoppedEv applies to CiscoCallID.getCall().
CiscoRTPHandle	getCiscoRTPHandle()	Returns a CiscoRTPHandle object. Applications can get a call reference by using CiscoProvider.getCall. If there is no call observer or there was no call observer when this event is delivered, CiscoProvider.getCall may return null.
int	getMediaConnectionMode()	• Returns CiscoMediaConnectionMode with one of the following values: • CiscoMediaConnectionMode. TRANSMIT_ONLY (one-way media; transmit) • onlyCiscoMediaConnectionMode. TRANSMIT_AND_RECEIVE (two-way media) **Note** In general, you should never get an event with mode NONE; however, if that happens, applications should ignore the event and log an error.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661, CiscoCallID, CiscoRTPParams, and CiscoMediaConnectionMode.

CiscoRTPPayload

The RTPPayload interface contains constants that describe RTP formats. The CiscoRTPInputProperties.getPayloadType and CiscoRTPOutputProperties.getPayloadType methods return these constants.

Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	Added H261, H263, H264, H264_SVC, T120, and H224 Methods.
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoRTPPayload

Fields

Table 192: Fields in CiscoRTPPayload

Interface	Fields	Description
static final int	NONSTANDARD	A nonstandard RTP payload
static final int	G711ALAW64K	G.711 64K a-law payload
static final int	G711ALAW56K	G.711 56K a-law payload
static final int	G711ULAW64K	G.711 64K u-law payload
static final int	G711ULAW56K	G.711 56K u-law payload
static final int	G722_64K	G.722 64K payload
static final int	G722_56K	G.722 56K payload
static final int	G722_48K	G.722 48K payload
static final int	G7231	G.723.1 payload
static final int	G728	G.728 payload
static final int	G729	G.729 payload
static final int	G729ANNEXA	G.729a payload
static final int	IS11172AUDIOCAP	IS11172AUDIOCAP payload

Interface	Fields	Description
static final int	IS13818AUDIOCAP	IS13818AUDIOCAP payload
static final int	ACY_G729AASSN	ACY_G729AASSN payload
static final int	DATA64	DATA64 payload
static final int	DATA56	DATA56 payload
static final int	GSM	GSM payload
static final int	ACTIVEVOICE	ACTIVEVOICE payload
static final int	WIDEBAND_256K	Wide_Band_256k payload

Methods

Table 193: Methods in CiscoRTPPayload

Interface	Method	Description
static final int	H261	The rtp payload type associated with the multimedia stream is H261.
static final int	H263	The rtp payload type associated with the multimedia stream is H263.
static final int	H264	The rtp payload type associated with the multimedia stream is H264.
static final int	H264_SVC	The rtp payload type associated with the multimedia stream is H264_SVC.
static final int	T120	The rtp payload type associated with the multimedia stream is T120.
static final int	H224	The rtp payload type associated with the multimedia stream is H224.

Related Documentation

See CiscoRTPInputProperties.getPayloadType(), CiscoRTPOutputProperties.getPayloadType(), and Constant Field Values, on page 1661.

CiscoRTPProperties

This interface contains the rtp properties of the multi media streams information.

Table 194: Interface History

Cisco Unified Communications Manager Release Number	Description
10.0(1)	New interface.

Declaration

public interface CiscoRTPProperties

Methods

Table 195: Methods in CiscoRTPProperties

Interface	Method	Description
InetAddress	getReceptionAddress()	Specifies the receiving IP address.
int	getReceptionPort()	Specifies the receiving port number.
InetAddress	getTransmissionAddress()	Specifies the transmitting IP address.
boolean	getTransmissionPort()	Specifies the transmitting port number.
int	getPayloadType()	Returns the payload format. The payload type can be: • CiscoRTPPayload.H261 = 100 • CiscoRTPPayload.H263_VIDEO = 101 • CiscoRTPPayload.VIDEO = 102 • CiscoRTPPayload.H264 = 103 • CiscoRTPPayload.H264_SVC = 104 • CiscoRTPPayload.T120 = 105 • CiscoRTPPayload.H224 = 106
int	getMaxBitRate()	Returns the maximum bit rate (bits per second), which is the max allowed video bit rate as negotiated by media layer. The value is the smaller of the region BW(Bandwidth) setting and BW requested by the endpoints.

CiscoSynchronousObserver

The Cisco JTAPI implementation is designed to allow applications to invoke blocking JTAPI methods such as Call.connect() and TerminalConnection.answer() from within their observer callbacks. This means that applications are not subject to the restrictions imposed by the JTAPI specification, which cautions applications against using JTAPI methods from within observer callbacks.

Normally, when an application adds a new observer to a JTAPI object, the Cisco JTAPI implementation creates an event queue and an accompanying worker thread to service the new observer. If the same observer is added to another object, its queue and thread are reused; in effect, every unique observer object has a single queue and worker thread. As noted, the advantage of this arrangement is that an application may invoke blocking JTAPI methods from within its observer callback. A subtle disadvantage, however, is that accessor methods such as Call.getConnections() and Connection.getState() may not return results that are consistent with events when invoked from within the observer callback.

For example, suppose that an application creates and connects a call from address "A" to address "B." If the application is observing address "A", it might reasonably expect that when it receives the CallActiveEv, the state of the call will be Call.ACTIVE. This is not necessarily true, because the worker thread that delivers events to the application is decoupled from the internal JTAPI thread that updates object states. In fact, if "B" rejects the call from "A, " the call object might be in either the Call.ACTIVE state or the Call.INVALID state, depending on the exact moment at which the worker thread delivers the CallActiveEv.

Many applications will not be adversely affected by this asychronous behavior. Applications that would benefit from a coherent call model during observer callbacks, however, can selectively disable the queueing logic of the Cisco JTAPI implementation. Applications that implement the CiscoSynchronousObserver interface on their observer objects declare that they want events to be delivered synchronously to its observers. Events delivered to synchronous observers will match the states of the call model objects queried from within the observer callback.

Objects that implement the CiscoSynchronousObserver interface may not invoke blocking JTAPI methods from within their event callbacks. The consequences of doing so are unpredictable, and may include deadlocking the JTAPI implementation. On the other hand, you may safely use the accessor methods of any JTAPI object, such as Call.getState() or Connection.getState(). Applications should avoid calling any interface that returns an array such as Terminal.getAddresses() in synchronous callbacks.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoSynchronousObserver

Fields

None

Methods

None

Related Documentation

None

CiscoTermActivatedEv

If a Terminal is observed and the restriction status changes to active, the system sends this event to the application.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoTermActivatedEv extends CiscoProvEv

Fields

Table 196: Fields in CiscoTermActivatedEv

Interface	Field	Description
static int	ID	None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 197: Methods in CiscoTermActivatedEv

Interface	Method	Description
javax.telephony.Terminal	getTerminal()	Returns the Terminal that is activated.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermButtonPressedEv

CiscoTermButtonPressedEv event is delivered on the TerminalObserver when a button is pressed on the Terminal. To receive this event, an application must set the filter using ciscoTermEvFilter.setButtonPressedEnabled(true). The button pressed events respond only to the numeric keypad button presses on the Terminal as listed in the constants in this interface: 0-9, *, #, A, B, C, and D.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermButtonPressedEv extends CiscoTermEv

Fields

Table 198: Fields in CiscoTermButtonPressedEv

Interface	Field
staticint	CHARA
staticint	CHARB
staticint	CHARC
staticint	CHARD
staticint	EIGHT
staticint	FIVE
staticint	FOUR
staticint	ID
staticint	NINE
staticint	ONE
staticint	POUND
staticint	SEVEN
staticint	SIX
staticint	STAR
staticint	THREE
staticint	TWO
staticint	ZERO

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 199: Methods in CiscoTermButtonPressedEv

Interface	Method	Description
int	getButtonPressed()	The button pressed on the Terminal.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoTermEvFilter and Constant Field Values, on page 1661.

CiscoTermConnMonitoringEndEv

The system delivers the CiscoTermConnMonitoringEndEv event to the call observer when monitoring stops on the call or when call is disconnected.

Interface History

Cisco Unified Communications Manager Release	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnMonitoringEndEv extends javax.telephony.events.TermConnEv

Fields

Table 200: Fields in CiscoTermConnMonitoringEndEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 201: CiscoTermConnMonitoringEndEv Methods

Interface	Method	Description
Int	getMonitorType()	Returns the type of monitoring. The return value is always CiscoCall.SILENT_MONITOR.

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoTermConnMonitoringStartEv

The system delivers the CiscoTermConnMonitoringStartEv event to the call observer when monitoring starts on the call.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnMonitoringStartEv extends javax.telephony.events.TermConnEv

Fields

Table 202: Fields in CiscoTermConnMonitoringStartEv

Interface	Field
staticfinal int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 203: CiscoTermConnMonitoringStartEv Methods

Interface	Method	Description
int	getMonitorType()	Returns the type of monitoring. The return value is always CiscoCall.Silent_Monitor.

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 for more information.

CiscoTermConnMonitorInitiatorInfoEv

The CiscoTermConnMonitorInitiatorInfoEv event interface extends the TermConnEv interface and gets reported via the CallObserver on the monitor target (agent). This interface gives information about the monitor initiator (supervisor) when a monitor session gets established.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.0(1)	A new API getTransactionID() will be exposed to retrieve the transaction ID.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnMonitorInitiatorInfoEv extends javax.telephony.events.TermConnEv

Fields

Table 204: Fields in CiscoTermConnMonitorInitiatorInfoEv

Interface	Fields
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 205: Methods in CiscoTermConnMonitorInitiatorInfoEv

Interface	Method	Description
CiscoMonitorInitiatorInfo	getCiscoMonitorInitiatorInfo()	Returns the Terminal name and Address of the monitor initiator.
int	getTransactionID()	Returns the transaction ID.

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermConnMonitorTargetInfoEv

The CiscoTermConnMonitorTargetInfoEv event interface extends the TermConnEv interface and gets reported via the CallObserver on monitor initiator. This interface provides information about the monitor target (agent) when a monitor session gets established.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.0(1)	A new getTransactionID() will be exposed to retrieve the transaction ID.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnMonitorTargetInfoEv extends javax.telephony.events.TermConnEv

Fields

Table 206: Fields in CiscoTermConnMonitorTargetInfoEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 207: Methods in CiscoTermConnMonitorTargetInfoEv

Interface	Method	Description
CiscoMonitorTargetInfo	getCiscoMonitorTargetInfo()	Returns the Terminal name and Address of the monitor target.
int	getTransactionID()	Returns the transaction ID.

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermConnPrivacyChangedEv

The system sends the CiscoTermConnPrivacyChangedEv event when the privacy status of a TeminalConnection changes. If privacy is active, the user cannot Barge into the Call.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoTermConnPrivacyChangedEv

Fields

Table 208: Fields in CiscoTermConnPrivacyChangedEv

Interface	Field
staticint	ID

Methods

Table 209: Methods in CiscoTermConnPrivacyChangedEv

Interface	Method	Description
javax.telephony. TerminalConnection	getTerminalConnection()	Returns the TerminalConnection where privacy changed. You can call getPrivacyStatus on the TerminalConnection to check its privacy status.

Related Documentation

See Constant Field Values, on page 1661 and CiscoTerminalConnection.getPrivacyStatus().

CiscoTermConnRecordingEndEv

The JTAPI delivers CiscoTermConnRecordingEndEv to the call observer when call recording stops.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnRecordingEndEv extends javax.telephony.events.TermConnEv

Fields

staticintID

Inherited Fields

Fields inherited from interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermConnRecordingStartEv

The JTAPI delivers CiscoTermConnRecordingStartEv to the call observer when call recording starts.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnRecordingStartEv extends javax.telephony.events.TermConnEv

Fields

Table 210: Fields in CiscoTermConnRecordingStartEv

Interface	Field
static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermConnRecordingTargetInfoEv

The JTAPI delivers CiscoTermConnRecordingTargetInfoEv to the call observer of the recording initiator.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnRecordingTargetInfoEv extends javax.telephony.events.TermConnEv

Fields

Table 211: Fields in CiscoTermConnRecordingTargetInfoEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN,

META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 212: Methods in CiscoTermConnRecordingTargetInfoEv

Interface	Method	Description
CiscoRecorderInfo	getCiscoRecorderInfo()	Returns CiscoRecorderInfo, which provides the terminal name and address of the recording device.

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and CiscoRecorderInfo.

CiscoTermConnRecordingFailedEv

The JTAPI delivers CiscoTermConnRecordingFailedEv to the call observer when call recording failed.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnRecordingFailedEv extends TermConnEv

Fields

Table 213: Fields in CiscoTermConnRecordingStartEv

Interface	Field
static int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermConnSelectChangedEv

The JTAPI sends CiscoTermConnSelectChangedEv when the call select status of a TerminalConnection changes, either by feature invocation or manually.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

Declaration

public interface CiscoTermConnSelectChangedEv extends javax.telephony.events.TermConnEv

Fields

Table 214: Fields in CiscoTermConnSelectChangedEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.TermConnEv

getTerminalConnection

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermCreatedEv

The JTAPI sends the CiscoTermCreatedEv event to the provider observer of the application when a CiscoTerminal gets added to the provider domain.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoTermCreatedEv extends CiscoProvEv

Fields

Table 215: Fields in CiscoTermEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 216: Methods in CiscoTermCreatedEv

Interface	Method	Description
javax.telephony.Terminal	getTerminal()	Returns the Terminal object for which this event was sent.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDataEv

The JTAPI sends the CiscoTermDataEv event to the terminal observer when the phone receives XSI data (XML object).

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDataEv extends CiscoTermEv

Fields

Table 217: Fields in CiscoTermDataEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 218: Methods in CiscoTermDataEv

Interface	Method	Description
java.lang.String	getData()	Deprecated.Use byte[] getTermData
byte[]	getTermData()	Returns an XML-encoded byte array corresponding to the XSI data that the phone received.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDeviceStateActiveEv

The CiscoTermDeviceStateActiveEv event gets sent to the Terminal Observer if any of the addresses on the terminal have an outgoing call in any state or an incoming call with TerminalConnection and CallCtlTerminalConnection in ACTIVE and TALKING state respectively.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDeviceStateActiveEv extends CiscoTermEv

Fields

Table 219: Fields in CiscoTermDeviceStateActiveEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDeviceStateAlertingEv

The CiscoTermDeviceStateAlertingEv event gets sent to the Terminal Observer if none of the Addresses on the Terminal have an outgoing call or an incoming call with Connection in CallCtlConnection.Established

state, and at least one of the addresses on the Terminal has an incoming Call with Connection in CallCtlConnection.OFFERED or CallCtlConnection.ALERTING State.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDeviceStateAlertingEv extends CiscoTermEv

Fields

Table 220: Fields in CiscoTermDeviceStateAlertingEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDeviceStateHeldEv

The CiscoTermDeviceStateHeldEv event gets sent to the Terminal Observer if all of the calls on the addresses of the Terminal have TerminalConnection in CallCtlTerminalConnection.HELD state.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDeviceStateHeldEv extends CiscoTermEv

Fields

Table 221: Fields in CiscoTermDeviceStateHeldEv

Interface	Field
staticint	ID

Inherited Fields

Fields Inherited From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDeviceStateIdleEv

The CiscoTermDeviceStateIdleEv event gets sent to the Terminal Observer if there are no calls on any of the Addresses of the Terminal.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDeviceStateIdleEv extends CiscoTermEv

Fields

Table 222: Fields in CiscoTermDeviceStateIdleEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDeviceStateWhisperEv

The CiscoTermDeviceStateActiveEv event gets sent to Terminal Observer if atleast one of the Addresses on the Terminal is an intercom target and has an intercom call with the TerminalConnection/CallCtlTerminalConneciton in State Passive/Bridged state. In this state, the user can initiate new outgoing calls and receive new incoming calls.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDeviceStateWhisperEv extends CiscoTermEv

Fields

Table 223: Fields in CiscoTermDeviceStateWhisperEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermDNDOptionChangedEv

The CiscoTermDNDOptionChangedEv event is delivered to the terminal observer if DND option is changed. This event is delivered only if the filter to receive events is enabled by the application. This event is provided on application observer

History

Cisco Unified Communications Manager Release	Description
7.0(1)	Added the extension.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

public interface CiscoTermDNDOptionChangedEv

extends CiscoTermEv

Fields

Table 224: CiscoTermDNDOptionChangedEv Fields

Interface	Field
static final int	ID

Table 225: Inherited Fields

From Interface javax.telephony.events.Ev
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Table 226: Inherited Fields

From Interface javax.telephony.events.Ev
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 227: CiscoTermDNDOptionChangedEv Methods

Interface	Method	Description
Int	getDNDOption()	This interface returns the current DND option to the application. It returns int dndOption.

Table 228: Inherited Methods

From Interface javax.telephony.events.Ev
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Table 229: Inherited Methods

From Interface javax.telephony.events.TermEv
getTerminal

Table 230: Inherited Methods

From Interface javax.telephony.events.Ev
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

See also Constant Field Values, on page 1661. and CiscoTermEv.

CiscoTermDNDStatusChangedEv

The CiscoTermDNDStatusChangedEv event gets delivered to the Terminal Observer if the DND status changes, provided that the application has enabled the filter to receive events.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermDNDStatusChangedEv extends CiscoTermEv

Fields

Table 231: Fields in CiscoTermDNDStatusChangedEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 232: Methods in CiscoTermDNDStatusChangedEv

Interface	Method	Description
boolean	getDNDStatus()	Returns the current DND status to the application.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoTermEvFilter, CiscoTermEv, and Constant Field Values, on page 1661.

CiscoTermEv

The CiscoTermEv interface, which extends the JTAPI core javax.telephony.events.TermEv interface, serves as the base interface for all Cisco-extended JTAPI Terminal events. Every Call-related event in this package extends this interface, directly or indirectly.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Subinterfaces

CiscoMediaOpenLogicalChannelEv, CiscoRTPInputKeyEv, CiscoRTPInputStartedEv, CiscoRTPInputStoppedEv, CiscoRTPOutputKeyEv, CiscoRTPOutputStartedEv, CiscoRTPOutputStoppedEv,

CiscoTermButtonPressedEv, CiscoTermDataEv, CiscoTermDeviceStateActiveEv, CiscoTermDeviceStateAlertingEv, CiscoTermDeviceStateHeldEv, CiscoTermDeviceStateIdleEv, CiscoTermDeviceStateWhisperEv, CiscoTermDNDOptionChangedEv, CiscoTermDNDStatusChangedEv, CiscoTermInServiceEv, CiscoTermOutOfServiceEv, CiscoTermRegistrationFailedEv, CiscoTermSnapshotCompletedEv, CiscoTermSnapshotEv

Declaration

public interface CiscoTermEv extends CiscoEv, javax.telephony.events.TermEv

Fields

None

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CallEv.

CiscoTermEvFilter

An application can use the CiscoTermEvFilter interface to selectively restrict those Terminal events that are not of interest.

Interface History

Cisco Unified Communications Manager Release Number	Description
11.5(1)	Added getHuntLogStatusChangedEvFilter() and setHuntLogStatusChangedEvFilter(boolean filterValue) methods.
10.0(1)	Added getMultiMediaStreamsInfoEvFilter() and setMultiMediaStreamsInfoEvFilter(boolean filterValue) methods.
7.1(1 and 2)	Created history table to track changes.

Declaration

public interface CiscoTermEvFilter

Fields

None

Methods

Table 233: Methods in CiscoTermEvFilter

Interface	Method	Description
boolean	getDeviceDataEnabled()	Returns the event filter status of the CiscoTermDataEv event for the Terminal. The default value is disabled. Returns True if the event filter is enabled, or false if the event filter is disabled.
void	setDeviceDataEnabled(booleanenabled)	Enables or disables the CiscoTermDataEv events for the Terminal.

Interface	Method	Description
boolean	getButtonPressedEnabled()	Returns the event filter status of the CiscoTermButtonPressedEv event for the Terminal. The default value is disabled. Returns:True if the event filter is enabled, or false if the event filter is disabled.
void	setButtonPressedEnabled(booleanenabled)	Enables or disables CiscoTermButtonPressedEv events for the Terminal.
boolean	getRTPEventsEnabled()	Returns the event filter status of the CiscoRTPInputStartedEv, CiscoRTPOutputStartedEv, CiscoRTPInputStoppedEv, and CiscoRTPOutputStoppedEv events for the Terminal. The Default value is disabled. Returns:True if the event filter is enabled, or false if the event filter is disabled.
void	setRTPEventsEnabled(booleanenabled)	Enables or disables CiscoRTPInputStartedEv, CiscoRTPOutputStartedEv, CiscoRTPInputStoppedEv and CiscoRTPOutputStoppedEv events for the Terminal.
boolean	getSnapshotEnabled()	Returns the event filter status of the CiscoTermSnapshotEv and CiscoTermSnapshotCompletedEv events for the Terminal. If disabled, neither event gets sent to applications. Returns:True if the event filter is enabled, or false if the event filter is disabled.
void	setSnapshotEnabled(booleanenabled)	Enable or disables CiscoTermSnapshotEv and CiscoTermSnapshotCompletedEv for the Terminal.
boolean	getRTPKeyEventsEnabled()	Returns the event filter status of the CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv events for the Terminal. Returns:True if the event filter is enabled, or false if the event filter is disabled.
void	setRTPKeyEventsEnabled(booleanenabled)	Enables or disables the CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv events for the Terminal.
boolean	getDeviceStateActiveEvFilter()	Returns the event filter status of the CiscoTermDeviceStateActiveEv event for the Terminal. Returns:True if the event filter is enabled, or false if the event filter is disabled.
boolean	getDeviceStateHeldEvFilter()	Returns the event filter status of the CiscoTermDeviceStateHeldEv event for the Terminal. Returns: True if the event filter is enabled, or false if the event filter is disabled.
boolean	getDeviceStateAlertingEvFilter()	Returns the event filter status of the CiscoTermDeviceStateAlerting event for the Terminal. Returns:True if the event filter is enabled, or false if the event filter is disabled.

Interface	Method	Description
boolean	getDeviceStateIdleEvFilter()	Returns the CiscoTermDeviceStateIdleEv filter status. Returns:True if the event filter is enabled, or false if the event filter is disabled.
void	setDeviceStateActiveEvFilter(booleanfilterValue)	Enables or disables the CiscoTermDeviceStateActiveEv filter for the Terminal. Default value is disable.
void	setDeviceStateHeldEvFilter(booleanfilterValue)	Enables or disables the CiscoTermDeviceStateHeldEv filter for the Terminal. Default value is disable
void	setDeviceStateAlertingEvFilter(booleanfilterValue)	Enables or disables the CiscoTermDeviceStateAlertingEv filter for the Terminal. Default value is disable
void	setDeviceStateIdleEvFilter(booleanfilterValue)	Enables or disables the CiscoTermDeviceStateIdleEv filter for the Terminal. Default value is disable
boolean	getDeviceStateWhisperEvFilter()	Returns the CiscoTermDeviceStateWhisperEv filter status on the Terminal.
boolean	getDNDChangedEvFilter()	Returns the CiscoTermDNDStatusChangedEv filter status Returns:the CiscoTermDNDStatusChangedEv Filter status on the Terminal.
void	setDNDChangedEvFilter(booleanfilterValue)	Enables or disables the CiscoTermDNDStatusChangedEv filter for the Terminal. Parameters:filterValue - void setDeviceStateWhisperEvFilter(booleanfilterValue) Enables or disables the CiscoTermDeviceStateWhisperEv filter for the Terminal. The default value is disable.
boolean	getDNDOptionChangedEvFilter()	This interface can be used to get CiscoTermDNDOptionChangedEv filter status . Returns:the CiscoTermDNDOptionChangedEv Filter status on the Terminal.
void	setDNDOptionChangedEvFilter(booleanfilterValue)	This interface is provided for enabling/disabling the CiscoTermDNDOptionChangedEv filter for the Terminal Parameters:filterValue.
boolean	getMultiMediaStreamsInfoEvFilter()	This interface can be used to get CiscoMultiMediaStreamsInfoEv filter status.
void	setMultiMediaStreamsInfoEvFilter(boolean filterValue)	This interface is provided for enabling/disabling the CiscoMultiMediaStreamsInfoEv filter for the Terminal.
boolean	getHuntLogStatusChangedEvFilter()	This method is used get the value of filter, the value of filter is false by default.

Interface	Method	Description
void	setHuntLogStatusChangedEvFilter(boolean filterValue)	This method is used to set the filter, if filter value is true, the event CiscoTermHuntLogStatusChangedEv is received by the application when the value of huntLogStatus is changed.

Related Documentation

None

CiscoTerminal

Standard JTAPI does not support the notion of dynamic terminal registration. The CiscoTerminal interface extends the standard terminal interface to do so. All Cisco Unified Communications Manager devices are represented by CiscoTerminals, and you can query all CiscoTerminals to determine whether they are currently IN_SERVICE or OUT_OF_SERVICE.

If the Cisco Unified Communications Manager device represented by the CiscoTerminal is an IP telephone, for instance, it goes OUT_OF_SERVICE if it loses its network connection. Other types of devices, such as CiscoMediaTerminal, get registered on demand by applications, and may be IN_SERVICE or OUT_OF_SERVICE accordingly.

CiscoTerminal includes an API getIPAddressingMode(). This interface returns the configured IP Addressing Mode of the CiscoTerminal.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
7.1.(3)	This interface is enhanced to: • Get the IP addresses of the terminal • Get the outbound call roll over configuration of the terminal. New interfaces expose consult call roll over, out bound call rollover, Join across lines (JAL) and Direct Transfer Across Lines (DTAL) capability of the terminal. A terminal can have different capability when feature is invoked on the phone and when feature is invoked by application. It does not indicate if the entire configuration required for the feature is enabled (for example, Join softkey must be configured in the template for the feature). • Get registered state of the terminal

Cisco Unified Communications Manager Release Number	Description
8.0(1)	Added the following new APIs: • APIs pickup(Address pickingAddress) groupPickup(Address pickingAddress • String pickupGroupNumber) • directedPickup(Address pickingAddress, String pickupGroupNumber), • Pickup(Address pickingAddress) for picking up calls from different flavors of Call Pickup Groups. • API getLoginType () which returns the LoginType on the terminal.
8.5(1)	A new API, isBuiltInBridgeEnabled(). is added.
10.0(1)	A new interface, public interface CiscoTerminal, is added on CiscoTerminal to determine the multimedia capabilities of the terminal.
11.5(1)	Added getHuntLogStatus() throwsInvalidStateException and setHuntLogStatus(int huntLogStatus)throws InvalidStateException,MethodNotSupportedException, methods.

Sample Code

```
public class MyTerminalObserver implements TerminalObserver
{
  public void terminalChangedEvent (TermEv[] evlist) {
    for(int i = 0; evlist ! =  null && i < evlist.length; i++)
    {
      …
      if ( evlist[i] instance of TermInServiceEv)
      {
        CiscoTerminal term  =  (CiscoTerminal)  (((CiscoTermEv)evlist[i]
).getTerminal());
        if(!term instanceof CiscoMediaTerminal && ! term instanmceof
CiscoRouteterminal)
        {
          try
          {
            if ( term. isBuiltInBridgeEnabled())
            {
              System.out.println("Build in Bridge is enabled for terminal" +
                tern.getName());
              else
              {
                System.out.println("Build in Bridge is disabled for terminal" +
                  tern.getName());
              }
            }
            catch(Exception)
            {
              System.out.println("Exception caught");
            }
          }
```

```
            }
        }
    }
}
```

Superinterfaces

CiscoObjectContainer, javax.telephony.Terminal

Subinterfaces

CiscoMediaTerminal, CiscoRouteTerminal

Declaration

public interface CiscoTerminal extends javax.telephony.Terminal, CiscoObjectContainer

Fields

Table 234: Fields in CiscoTerminal

Interface	Field	Description
static final int	DEVICE_HUNT_LOGGED_IN	This constant depicts that the terminal is logged into the hunt group.
static final int	DEVICE_HUNT_LOGGED_OUT	This constant depicts that the terminal is logged out of the hunt group.
static final int	DEVICE_HUNT_NOT_APPLICABLE	This constant depicts that the terminal does not have the capability either to log in or log out of the hunt group.
static final int	OUT_OF_SERVICE	This Constant Field Values, on page 1661 returned by the getState() interface on CiscoTerminal indicates that the CiscoTerminal is out of service.
static final int	IN_SERVICE	This constant value returned by the getState() interface on CiscoTerminal indicates that the CiscoTerminal is in service.
static final int	DEVICESTATE_IDLE	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal currently has no calls on any Addresses.
static final int	DEVICESTATE_ACTIVE	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal has at least one active call on an Address.

Interface	Field	Description
static final int	DEVICESTATE_ALERTING	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal has at least one alerting call, but no active call, on an Address.
static final int	DEVICESTATE_HELD	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal has at least one held call, but no alerting or active calls, on an Address.
static final int	DEVICESTATE_UNKNOWN	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal DeviceState is Unknown. This state may get returned if the device state filters are disabled.
static final int	DEVICESTATE_WHISPER	This constant value returned by the getDeviceState() interface on CiscoTerminal indicates that the CiscoTerminal has at least one intercom call with one-way media, but has no held, alerting, or active calls on an Address.
static final int	UNKNOWN_ENCODING	Indicates that the CiscoTerminal.getSupportedEncoding () for this terminal is UNKNOWN.
static final int	NOT_APPLICABLE	Indicates that the CiscoTerminal.getSupportedEncoding () for this CiscoMediaTerminal or RoutePoint is NOT_APPLICABLE.
static final int	ASCII_ENCODING	Indicates that the CiscoTerminal.getSupportedEncoding () for this terminal is ASCII and that this terminal supports only ASCII_ENCODING.
static final int	UCS2UNICODE_ENCODING	Indicates that the CiscoTerminal.getSupportedEncoding () for this terminal is UCS2UNICODE_ENCODING.
static final int	DND_OPTION_NONE	This constant value returned by the getDNDOption() interface on CiscoTerminal indicates that the DND option configured is None.
static final int	DND_OPTION_RINGER_OFF	This constant value returned by the getDNDOption() interface on CiscoTerminal indicates that the DND option configured is Ringer Off. If DND is enabled on the phone, the phone would not ring if a call lands there.

Interface	Field	Description
static final int	DND_OPTION_CALL_REJECT	This constant value returned by the getDNDOptions() interface on CiscoTerminal indicates that the DND option configured is Call Reject. If DND is enabled on the phone, all calls to the phone will get rejected, except for shared lines.
static final int	IP_ADDRESSING_MODE_UNKNOWN	This indicates that IPAddressing mode is Unknown.
static final int	IP_ADDRESSING_MODE_IPV4	This indicates that IPAddressing mode is IPv4
static final int	IP_ADDRESSING_MODE_IPV6	This indicates that IPAddressing mode is IPv6
static final int	IP_ADDRESSING_MODE_IPV4_V6	This indicates that IPAddressing mode is both IPv4 and IPv6
static final int	IP_ADDRESSING_MODE_UNKNOWN_ANATRED	This is reserved IP Addressing constant for ANAT in Cisco Unified Communications Manager. It is not used in JTAPI

Methods

Table 235: Methods in CiscoTerminal

Inteface	Method	Description
int	getLoginType()	This method returns the value NO_EM_LOGIN, NATIVE_LOGIN or VISITOR_LOGIN to indicate whether the terminal is local to the cluster or not when the EM login is done.
Static final int	CiscoTerminal.NO_LOGIN	This indicates that there has been no EM login done into the terminal. It will have an integer value of 0.
	CiscoTerminal.NATIVE_LOGIN	This indicates that the terminal is part of the local cluster when an EM login is done into it with a profile that belongs to the same cluster. It will have an inteher value of 1.
	CiscoTerminal.VISITOR_LOGIN	This indicates that the terminal is part of the visiting cluster when an EM login is done into it with a profile that is not local to the cluster. It will have an integer value of 2.
int	getRegistrationState()	Deprecated This method has been replaced by the getState() method. Returns the state of this terminal. The state may be any of the following constants: • CiscoTerminal.OUT_OF_SERVICE • CiscoTerminal.IN_SERVICE

Inteface	Method	Description
int	getState()	Returns the state of this terminal. The state may be any of the following constants: • CiscoTerminal.OUT_OF_SERVICE • CiscoTerminal.IN_SERVICE
CiscoRTPInputProperties	getRTPInputProperties()	Returns the properties to be used for the RTP input stream associated with the ACTIVE TerminalConnection on this terminal. The CiscoTerminal must be in the CiscoTerminal.REGISTERED state, its Provider must be in the Provider.IN_SERVICE state, and Terminal.getTerminalConnections () must return at least one terminal connection in the TerminalConnection.ACTIVE state. **Throws** javax.telephony.InvalidStateException
CiscoRTPOutputProperties	getRTPOutputProperties()	Returns the properties to be used for the RTP output stream associated with the ACTIVE TerminalConnection on this terminal. The CiscoTerminal must be in the CiscoTerminal.REGISTERED state, its Provider must be in the Provider.IN_SERVICE state, and Terminal.getTerminalConnections () must return at least one terminal connection in the TerminalConnection.ACTIVE state. **Throws** javax.telephony.InvalidStateException
java.lang.String	sendData(java.lang. StringterminalData)	Deprecated Use CiscoTerminal.sendData (byte[]). **Throws** javax.telephony.InvalidStateException javax.telephony.MethodNotSupportedException
byte[]	sendData(byte[]terminalData)	The CiscoTerminal must be in the CiscoTerminal.IN_SERVICE state, and its Provider must be in the Provider.IN_SERVICE state. Applications can push the XSI object in the byte format to the phone with this interface. If the phone receives the data, this method returns successfully. Applications may only send 2000 bytes of data with this interface. Requests carrying excess data get rejected. **Throws** PlatformException (The data did not get sent successfully), javax.telephony.InvalidStateException, and javax.telephony.MethodNotSupportedException
CiscoTermEvFilter	getFilter()	Retrieves the filter object associated with the terminal.

Inteface	Method	Description
void	setFilter(CiscoTermEvFilter terminalEvFilter)	Filters the events that get delivered to the TerminalObserver. You can call this method at any time, but the typical usage is to set up the terminal events as part of initialization or when a CiscTermCreatedEv indicates that the system created a new terminal. • Example 1—One use might be to turn on the button-pressed events that normally do not get not delivered. Terminal term = provider.getTerminal (name); if (term instanceof CiscoTerm) { CiscoTerm ciscoTerm = (CiscoTerm)term; CiscoTermEvFilter filter = ciscoTerm.getFilter (); filter.setButtonPressedEnabled (true); } term.addObserver (terminalObserver) • Example 2—Another use might be turning off events that are not of interest to an application. For example, an application doing pure call control could turn off the media (RTP) events as follows: Terminal term = provider.getTerminal (name); if (term instanceof CiscoTerm) { CiscoTerm ciscoTerm = (CiscoTerm)term; CiscoTermEvFilter filter = ciscoTerm.getFilter (); filter.setRTPEventsEnabled (false); ciscoTerm.setFilter (filter); } term.addObserver (terminalObserver); term.getAddresses () [0].addCallObserver (callObserver) **Note** Adding a CallObserver (without explicitly setting a filter) turns the RTP events on. This behavior of Cisco JTAPI Release 1.4 and earlier is still preserved. If an explicit setFilter call gets made, the filter settings will take effect. The RTP events will not get delivered for the previous code snippet, but will get delivered for the following example: Terminal term = provider.getTerminal (name); term.addObserver (terminalObserver); term.getAddresses () [0].addCallObserver (callObserver).

Inteface	Method	Description
javax.telephony.TerminalConnection	unPark(javax.telephony.Address UnParkAddress, java.lang.StringParkedAt)	Returns a terminalConnection. The CiscoTerminal must be in the CiscoTerminal.IN_SERVICE state and its Provider must be in the Provider.IN_SERVICE state. This method takes an address and string as input. **Parameters** • UnParkAddress - Any address on the terminal • ParkedAt - A string identifying is system park port where a call was previously parked. The system returns this string when the call gets parked. **Throws** • javax.telephony.InvalidStateException (The CiscoTerminal.getState() is not IN_SERVICE) • PlatformException - Any other error occurred while unparking (for example, the Unpark number is not valid). • javax.telephony.InvalidArgumentException • javax.telephony.ResourceUnavailableException
int	getDeviceState()	• Returns the DeviceState of this terminal. The DeviceState is the accumulative call state of all the addresses on the terminal. The state may be any of the following constants: • CiscoTerminal.DEVICESTATE_ILDE • CiscoTerminal.DEVICESTATE_ACTIVE • CiscoTerminal.DEVICESTATE_ALERTING • CiscoTerminal.DEVICESTATE_HELD • CiscoTerminal.DEVICESTATE_UNKNOWN • CiscoTerminal.DEVICESTATE_WHISPER **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.

Inteface	Method	Description
int	getSupportedEncoding()	Returns the supported encoding types for this terminal. Use this method to check whether a terminal supports Unicode. To access this information, the terminal must be in the CiscoTerminal.IN_SERVICE state. The supportedEncoding is one of the following constants: • CiscoTerminal.UNKNOWN_ENCODING • CiscoTerminal.ASCII_ENCODING • CiscoTerminal.UCS2UNICODE_ENCODING • CiscoTerminal.NOT_APPLICABLE **Throws** • javax.telephony.InvalidStateException
int	getLocale()	Returns the locale that this terminal supports. To access this method, the terminal must be in the CiscoTerminal.IN_SERVICE state. **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.
boolean	isRestricted()	Returns the restriction status of this terminal. If a terminal is restricted, all associated addresses on the terminal are also restricted. Returns:True if terminal is restricted; otherwise false.
void	createSnapshot()	Generates a CiscoTermSnapshotEv event, which contains the security status of the current active call on the terminal. To access this method, the terminal must be in the CiscoTerminal.IN_SERVICE state and CiscoTermEvFilter.setSnapshotEnabled () must be set to true. **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.
java.lang.String	getAltScript()	Returns the locale alternate script that this terminal supports. An empty return value indicates that this terminal does not support or is not configured with an alternate script. To access this method, the terminal must be in the CiscoTerminal.IN_SERVICE state. **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.

Inteface	Method	Description
int	getProtocol()	Reports the terminal protocol (SCCP, SIP, or none) and returns the protocol of this terminal as one of the following constants: • CiscoTerminalProtocol.PROTOCOL_NONE • CiscoTerminalProtocol.PROTOCOL_SCCP • CiscoTerminalProtocol.PROTOCOL_SIP
void	setDNDStatus(booleandndStatus)	Sets the DND status, which enables or disables the DND feature. This feature does not apply to route points. **Parameters** • dndStatus **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.
boolean	getDNDStatus()	Reports the DND status and returns dndStatus. **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.
int	getDNDOption()	Returns the value of the DND option. This value is not significant for a CiscoMediaTerminal or CiscoRouteTerminal because the DND feature applies only to physical phones. The DND option can be any of the following constants: • CiscoTerminal.DND_OPTION_NONE • CiscoTerminal.DND_OPTION_RINGER_OFF • CiscoTerminal.DND_OPTION_CALL_REJECT **Throws** • javax.telephony.InvalidStateException —CiscoTerminal.getState() is not IN_SERVICE.
java.lang.String	getEMLoginUsername()	Returns extension mobility (EM) login user name. If no EM user has logged into Terminal, this interface will return null/empty string **Throws** • javax.telephony.InvalidStateException —if CiscoTerminal.getState() is not IN_SERVICE. **Note** You must use this API with CiscoTerminal.getLoginType() to determine if an EM login is done.

Inteface	Method	Description
InetAddress	getIPV4Address()	Returns IPV4 ip address of the terminal **Throws** • InvalidStateException • MethodNotSupportedException
InetAddress	getIPV6Address()	Returns IPV6 ip address of the terminal **Throws** • InvalidStateException • MethodNotSupportedException
Call	pickup (Address pickingAddress)	This method picks up the longest ringing call from the Pickup Group to which pickingAddress belongs to. pickingAddress is the Address on the Terminal where the Call is picked up. **Parameters** pickingAddress that specifies which Address the Terminal object should do the pickup on.
Call	groupPickup(Address pickingAddress, String pickupGroupNumber)	This method picks up the longest ringing call from the specified pickupGroupNumber at the pickingAddress. **Parameters** • pickingAddress that specifies which Address the Terminal object should do the pickup on. • Additional String object that respresents the number of the pickup group you wish to answer a call from, as set in the CUCM.
Call	directedPickup(Address pickingAddress, String ringingDN)	This method picks up the call from the specified ringingDN at the pickingAddress. The ringingDN must be in the same Pickup Group as the pickingAddress. **Parameters** • pickingAddress that specifies which Address the Terminal object should do the pickup on. • Additional String object that respresents the specific DN the application wishes to pick up for a directed call pickup request.

Inteface	Method	Description
Call	otherPickup(Address pickingAddress)	This method picks up a call based upon the priority of the associated pickupgroups. Within the group, if there are more than one call, the longest ringing call will be picked up. pickingAddress is the Address on the Terminal where the Call is picked up. **Parameters** pickingAddress that specifies which Address the Terminal object should do the pickup on.
int	getRollOverConfig()	Returns the roll over configuration of the terminal. **Throws** • InvalidStateException.
public final static int	NO_ROLLOVER	This indicates that the terminal is configured with no roll over.
public final static int	ROLLOVER_ANY_DN	This indicates that calls can roll over to any address on the terminal.
public final static int	ROLLOVER_SAME_DN	This indicates that calls can roll over to address that match the name(DN).
public static final int	PHONE_USER	Application can use this to find the capability of the terminal when used manually by user.
publis static final int	APPLICATION	Application can use to this to find the capability of the terminal to invoke the feature from application.
boolean	canConsultCallRollOver(int which_user)	Which user can be CiscoTerminal. PHONE_USER or CiscoTerminal.APPLICATION **Throws** • InvalidStateException.
boolean	canOutBoundCallRollOver(int which_user)	• InvalidStateException.
boolean	canJoinAcrossLines(int which_user)	This interface returns True if calls on different addresses of this terminal can be conferenced. This interface returns True for terminals that support Connected Transfer or Conference Across Lines. **Throws** • InvalidStateException.

Inteface	Method	Description
boolean	canDirectTransferAcrossLines(int which_user)	This interface returns True if calls on different addresses of this terminal can be transferred. This interface returns True for terminals that support Connected Transfer or Conference Across Lines. **Throws** • InvalidStateException.
boolean	canJoinOnSameLine (int which_user)	**Throws** • InvalidStateException.
boolean	canDirectTransferOnSameLine(int which_user)	**Throws** • InvalidStateException.
boolean	isRegistered()	Returns true if terminal is registered with Cisco Unified Communications Manager else false.
public boolean	isBuiltInBridgeEnabled()	Returns true if Built-In-Bridge is enabled on the terminal, else returns false.
void	register()	This method registers Cisco Unified Client Services Framework device to Extend mode, which will be represented as a CiscoRemoteTerminal. This is intended to be used by application monitoring Cisco Unified Client Services Framework device to enable its Extend mode from its softphone (SIP)/deskphone mode. The successful effect of this method is to register the device and present as a CiscoRemoteTerminal terminal type (if it is not already a CiscoRemoteTerminal), and be able to use Cisco Extend & Connect (CTI Remote Device) supported features with remote destinations. In any case when it switches in between Softphone/Deskphone & Extend modes that results in a terminal switching (e.g. CiscoTerminal->CiscoRemoteTerminal), there will be provider events sent to application: CiscoAddrRemovedEv, CiscoTermRemovedEv, CiscoAddrCreatedEv, CiscoTermCreatedEv. Any observers on the terminal and address will be removed, application needs to de-reference the old terminal object and re-add any desired observers. Note that CiscoProvider must be in IN_SERVICE state, otherwise CiscoRegistrationException about InvalidStateException will be thrown; or if terminal is already registered by this application in Extend mode or the registration fails for any reason, CiscoRegistrationException will be thrown. And if terminal type is not CiscoTerminal or CiscoRemoteTerminal and if terminal is not a Cisco Unified Client Services Framework device, then MethodNotSupportedException will be thrown.

Inteface	Method	Description
void	unregister()	This method unregisters Cisco Unified Client Services Framework device from Extend mode. Its terminal type will remain as CiscoRemoteTerminal, and application can explicitly register to CUCM to switch back to its softphone/deskphone mode. This is intended to be used by application monitoring Cisco Unified Client Services Framework device in Cisco Extend Connect mode to disable its Cisco Extend Connect mode. The successful effect of this method is to unregister the device but retains as a CiscoRemoteTerminal. Note that CiscoProvider must be in IN_SERVICE state, otherwise CiscoUnregistrationException about InvalidStateException will be thrown; or if terminal is not registered in Extend mode by this application or the unregistration fails for any reason, CiscoUnregistrationException will be thrown. And if terminal type is not CiscoRemoteTerminal and if terminal is not a Cisco Unified Client Services Framework device, then MethodNotSupportedException will be thrown.

Inteface	Method	Description
int	getType()	This method returns the device type of Terminal, which can be one of the following contants. It will return CiscoTerminal.DEVICETYPE_UNKNOWN if type of device cannot be found or device is invalid. CiscoTerminal.DEVICETYPE_UNKNOWN CiscoTerminal.DEVICETYPE_ANALOG_PHONE CiscoTerminal.DEVICETYPE_CISCO_6901 CiscoTerminal.DEVICETYPE_CISCO_6911 CiscoTerminal.DEVICETYPE_CISCO_6921 CiscoTerminal.DEVICETYPE_CISCO_6941 CiscoTerminal.DEVICETYPE_CISCO_6945 CiscoTerminal.DEVICETYPE_CISCO_6961 CiscoTerminal.DEVICETYPE_CISCO_7906 CiscoTerminal.DEVICETYPE_TELECASTER_BID CiscoTerminal.DEVICETYPE_CISCO_7911 CiscoTerminal.DEVICETYPE_14 _BUTTON_SIDECAR CiscoTerminal.DEVICETYPE_7915_12 _BUTTON_SIDECAR CiscoTerminal.DEVICETYPE_7915_24 _BUTTON_SIDECAR CiscoTerminal.DEVICETYPE_7916_12 _BUTTON_SIDECAR CiscoTerminal.DEVICETYPE_7916_24 _BUTTON_SIDECAR CiscoTerminal.DEVICETYPE_CKEM_36_BUTTON CiscoTerminal.DEVICETYPE_CP7921 CiscoTerminal.DEVICETYPE_CISCO_7925 CiscoTerminal.DEVICETYPE_CISCO_7926 CiscoTerminal.DEVICETYPE_7931

Inteface	Method	Description
		CiscoTerminal.DEVICETYPE_IP_ CONFERENCE_PHONE CiscoTerminal.DEVICETYPE_CISCO_7936 CiscoTerminal.DEVICETYPE_CISCO_7937 CiscoTerminal.DEVICETYPE _TELECASTER_BUSINESS CiscoTerminal.DEVICETYPE_CISCO_7941 CiscoTerminal.DEVICETYPE_CISCO_7941G_GE CiscoTerminal.DEVICETYPE_CISCO_7942 CiscoTerminal.DEVICETYPE_CISCO_7945 CiscoTerminal.DEVICETYPE_TELECASTER_MGR CiscoTerminal.DEVICETYPE_CISCO_7961 CiscoTerminal.DEVICETYPE_CISCO_7961G_GE CiscoTerminal.DEVICETYPE_CISCO_7962 CiscoTerminal.DEVICETYPE_CISCO_7965 CiscoTerminal.DEVICETYPE_CISCO_7970 CiscoTerminal.DEVICETYPE_CISCO_7971 CiscoTerminal.DEVICETYPE_CISCO_7975 CiscoTerminal.DEVICETYPE_CISCO_7989 CiscoTerminal.DEVICETYPE_CISCO_8941 CiscoTerminal.DEVICETYPE_CISCO_8945 CiscoTerminal.DEVICETYPE_CISCO_8961 CiscoTerminal.DEVICETYPE_9951 CiscoTerminal.DEVICETYPE_CISCO_9971 CiscoTerminal.DEVICETYPE_ATA_186 CiscoTerminal.DEVICETYPE_CISCO_ATA_187 CiscoTerminal.DEVICETYPE_CISCO_CIUS CiscoTerminal.DEVICETYPE_CISCO_CIUS_SP CiscoTerminal.DEVICETYPE_CISCO _SOFTPHONE_SE_M CiscoTerminal.DEVICETYPE_CISCO_UNIFIED _COMMUNICATOR CiscoTerminal.DEVICETYPE_CISCO_UNIFIED _MOBILE_COMMUNICATOR CiscoTerminal.DEVICETYPE_CISCO_UNIFIED _COMMUNICATIONS_FOR_RTX

Inteface	Method	Description
		CiscoTerminal.DEVICETYPE_CLIENT _SERVICES_FRAMEWORK CiscoTerminal.DEVICETYPE_VGC_PHONE CiscoTerminal.DEVICETYPE_CTI_PORT CiscoTerminal.DEVICETYPE_CTI_ROUTE_POINT CiscoTerminal.DEVICETYPE_DEVICE_PILOT CiscoTerminal.DEVICETYPE_ISDN_BRI_PHONE CiscoTerminal.DEVICETYPE_CTI_REMOTE_DEVICE
String	getTypeName()	This method returns the device type name of Terminal (i.e. The display name of the device type). It will return an empty string if device type name cannot be found, or return null if device is invalid.
CiscoTerminal	getCiscoMultiMediaCapabilityInfo()	Returns CiscoMultiMediaCapabilityInfo, to indicate the multimedia capabilities of the terminal.
int	getHuntLogStatus() throws InvalidStateException	This method is used get the value of huntlogstatus of the terminal, it returns either CiscoTerminal.DEVICE_HUNT_LOGGED_IN, CiscoTerminal.DEVICE_HUNT_LOGGED_OUT or CiscoTerminal. DEVICE_HUNT_NOT_APPLICABLE This method throws InvalidStateException if it is invoked on the terminal which is out of service
void	setHuntLogStatus(int huntLogStatus) throws InvalidStateException, MethodNotSupportedException, InvalidArgumentException	This method is used set the value of huntLogStatus of the device, it can take CiscoTerminal.DEVICE_HUNT_LOGGED_IN, CiscoTerminal.DEVICE_HUNT_LOGGED_OUT values as parameters. This method throws InvalidStateException if it is invoked on the terminal which is out of service. It throws MethodNotSupportedException when it is invoked on unsupported devices like CTI Route Points, CTI Remote Device and Spark Remote Device and InvalidArgumentException when the arguments are other than 1(loggen_in) and 2(logged_out)

Inherited Fields

From Interface javax.telephony.Terminal

addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

Data Type

public static final int

CiscoTerminal.DEVICETYPE_UNKNOWN = 0

CiscoTerminal.DEVICETYPE_ANALOG_PHONE = 30027

CiscoTerminal.DEVICETYPE_12S = 4

CiscoTerminal.DEVICETYPE_CISCO_6901 = 547

CiscoTerminal.DEVICETYPE_CISCO_6911 = 548

CiscoTerminal.DEVICETYPE_CISCO_6921 = 495

CiscoTerminal.DEVICETYPE_CISCO_6941 = 496

CiscoTerminal.DEVICETYPE_CISCO_6945 = 564

CiscoTerminal.DEVICETYPE_CISCO_6961 = 497

CiscoTerminal.DEVICETYPE_CISCO_7906 = 369

CiscoTerminal.DEVICETYPE_TELECASTER_BID = 6

CiscoTerminal.DEVICETYPE_CISCO_7911 = 307

CiscoTerminal.DEVICETYPE_14_BUTTON_SIDECAR = 124

CiscoTerminal.DEVICETYPE_7915_12_BUTTON_SIDECAR = 227

CiscoTerminal.DEVICETYPE_7915_24_BUTTON_SIDECAR = 228

CiscoTerminal.DEVICETYPE_7916_12_BUTTON_SIDECAR = 229

CiscoTerminal.DEVICETYPE_7916_24_BUTTON_SIDECAR = 230

CiscoTerminal.DEVICETYPE_CKEM_36_BUTTON = 232

CiscoTerminal.DEVICETYPE_CP7921 = 365

CiscoTerminal.DEVICETYPE_CISCO_7925 = 484

CiscoTerminal.DEVICETYPE_CISCO_7926 = 577

CiscoTerminal.DEVICETYPE_7931 = 348

CiscoTerminal.DEVICETYPE_IP_CONFERENCE_PHONE = 9

CiscoTerminal.DEVICETYPE_CISCO_7936 = 30019

CiscoTerminal.DEVICETYPE_CISCO_7937 = 431

CiscoTerminal.DEVICETYPE_TELECASTER_BUSINESS = 8

CiscoTerminal.DEVICETYPE_CISCO_7941 = 115

CiscoTerminal.DEVICETYPE_CISCO_7941G_GE = 309

CiscoTerminal.DEVICETYPE_CISCO_7942 = 434

CiscoTerminal.DEVICETYPE_CISCO_7945 = 435

CiscoTerminal.DEVICETYPE_TELECASTER_MGR = 7

CiscoTerminal.DEVICETYPE_CISCO_7961 = 30018

CiscoTerminal.DEVICETYPE_CISCO_7961G_GE = 308

CiscoTerminal.DEVICETYPE_CISCO_7962 = 404

CiscoTerminal.DEVICETYPE_CISCO_7965 = 436

CiscoTerminal.DEVICETYPE_CISCO_7970 = 30006

CiscoTerminal.DEVICETYPE_CISCO_7971 = 119

CiscoTerminal.DEVICETYPE_CISCO_7975 = 437

CiscoTerminal.DEVICETYPE_CISCO_7989 = 302

CiscoTerminal.DEVICETYPE_CISCO_8941 = 586

CiscoTerminal.DEVICETYPE_CISCO_8945 = 585

CiscoTerminal.DEVICETYPE_CISCO_8961 = 540

CiscoTerminal.DEVICETYPE_9951 = 537

CiscoTerminal.DEVICETYPE_CISCO_9971 = 493

CiscoTerminal.DEVICETYPE_ATA_186 = 12

CiscoTerminal.DEVICETYPE_CISCO_ATA_187 = 550

CiscoTerminal.DEVICETYPE_CISCO_CIUS = 593

CiscoTerminal.DEVICETYPE_CISCO_CIUS_SP = 632

CiscoTerminal.DEVICETYPE_CISCO_SOFTPHONE_SE_M = 30016

CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_COMMUNICATOR = 358

CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_MOBILE_COMMUNICATOR = 468

CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_COMMUNICATIONS_FOR_RTX = 648

CiscoTerminal.DEVICETYPE_CLIENT_SERVICES_FRAMEWORK = 503

CiscoTerminal.DEVICETYPE_VGC_PHONE = 10

CiscoTerminal.DEVICETYPE_CTI_PORT = 72

CiscoTerminal.DEVICETYPE_CTI_ROUTE_POINT = 73

CiscoTerminal.DEVICETYPE_DEVICE_PILOT = 71

CiscoTerminal.DEVICETYPE_ISDN_BRI_PHONE = 30028

CiscoTerminal.DEVICETYPE_CTI_REMOTE_DEVICE = 635

Related Documentation

See Terminal, CiscoMediaTerminal, Constant Field Values, on page 1661, and CiscoTermEvFilter.

CiscoTerminalConnection

The CiscoTerminalConnection interface extends the CallControlTerminalConnection interface with additional capabilities. Applications can use the getReason method to obtain the reason for the creation of this Connection.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.5(1)	Two new methods, addMediaStream(String streamDN, String callingPartyNumber) and removeMediaStream(), are added.
	A new API, playtone(int toneType, int playToneDirection) is added.
9.0(1)	A new method, startRecording(int playToneDirection, int recordingInvocationType), is added. Two new constants, RECORDING_INVOCATION_TYPE_SILENT, and RECORDING_INVOCATION_TYPE_USER, are added.
10.0(1)	A new method, hold (String contentID) is added.

Superinterfaces

javax.telephony.callcontrol.CallControlTerminalConnection, CiscoObjectContainer, javax.telephony.TerminalConnection

Declaration

public interface CiscoTerminalConnection extends javax.telephony.callcontrol.CallControlTerminalConnection, CiscoObjectContainer

Fields

Table 236: Fields in CiscoTerminalConnection

Interface	Field	Description
static final int	CISCO_SELECTEDNONE	The the call is not selected.
static final int	CISCO_SELECTEDLOCAL	The call is selected.
static final int	CISCO_SELECTEDREMOTE	A passive TerminalConnection receives this select status if the call is selected by its shared line.

Interface	Field	Description
static int	RECORDING_INVOCATION_TYPE_SILENT	This constant is used when the application invokes silent recording invocation type. The call recording status is not reflected on the Cisco IP device display. Silent recording is the default behavior in releases prior to Release 9.0. If an application uses the startRecording(int playToneDirection) method that was introduced prior to Release 9.0, it will default to the RECORDING_INVOCATION_TYPE_SILENT invocation type.
static int	RECORDING_INVOCATION_TYPE_USER	This constant is used when the application invokes user recording invocation type. The call recording status is reflected on the Cisco IP device display. Applications can query the device for this recording type.

Inherited Fields

From Interface javax.telephony.callcontrol.CallControlTerminalConnection

BRIDGED, DROPPED, HELD, IDLE, INUSE, RINGING, TALKING, UNKNOWN

From Interface javax.telephony.TerminalConnection

ACTIVE, PASSIVE

Parameters

invocationType

The invocationType parameter allows an application to specify a recording invocation type. The parameter is passed as one of the constants RECORDING_INVOCATION_TYPE_SILENT or RECORDING_INVOCATION_TYPE_USER.

String contentID

A String representing a specific video. Max 128 characters.

New Error Codes

CTIERR_ILLEGAL_CALLSTATE

Occurs if the request is made on a TerminalConnection associated with an invalid call. The only valid state to invoke this request is ‘Connected’.

JTAPI throws InvalidStateException with description as “Line is not in a legal state to invoke command.”

CTIERR _CALL_DROPPED

Occurs if the request is made on a TerminalConnection associated with an invalid call.

JTAPI throws InvalidStateException with description as “Call is dropped”.

CTIERR_BIB_NOT_CONFIGURED

Occurs if the Built-In-Bridge (BIB) is not configured on the agent device.

JTAPI throws ResourceUnavailableException with a description as “Built in bridge not configured”.

CTIERR_RESOURCE_NOT_AVAILABLE

Occurs if the Bulit-In-Bridge (BIB) cannot be allocated for the request.

JTAPI throws ResourceUnavailableException with a description as “Resource Not Available”.

CTIERR_MEDIA_CONNECTION_FAILED

Occurs if the Bulit-In-Bridge (BIB) call fails to connect to the media.

JTAPI throws InvalidStateException with a description as “The connection to the media has failed”.

CTIERR_START_STREAM_MEDIA_FAILED

Occurs if there is a general failure with the Agent Greeting feature, that is not covered by any of the other error codes.

JTAPI throws InvalidStateException with a description as “Start streaming media request failed”.

CTIERR_STOP_STREAM_MEDIA_FAILED

Occurs if there is a general failure with the Agent Greeting feature, that is not covered by any of the other error codes.

JTAPI throws InvalidStateException with a description as “Stop streaming media request failed”.

CTIERR_REQUEST_ALREADY_PENDING

Occurs if an application attempts to invoke an Agent Greeting API while another request is made.

JTAPI throws InvalidStateException with a description as “The request was rejected because there is a similar request already pending”.

CTIERR_NO_STREAMING_MEDIA_SESSION

Occurs if an application attempts to invoke a stop request while there is no existing media stream to stop.
JTAPI throws InvalidStateException with a description as “There is no streaming media session active”.

CTIERR_EXISTING_STREAMING_MEDIA_SESSION

Occurs if an application attempts to invoke an Agent Greeting API while another request is made and accepted.

JTAPI throws InvalidStateException with a description as “There is an existing streaming media session”.

CTIERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING

Occurs if an application attempts to stop an active recording, but specifies a recording type other than the recording type that was used to invoke the recording.

Methods

Table 237: Methods in CiscoTerminalConnection

Interface	Method	Description
boolean	getPrivacyStatus()	Returns the privacy status of the call on the terminal. This interface returns True if Privacy is on and False otherwise. Always check the TerminalConnection privacy status before displaying any information about the call at an applcation Terminal implementation.
int	getSelectStatus()	Returns the select status of the call on the terminal. Always check the select status of the TerminalConnection before performing any call-process operation for the call. Can be one of: • CiscoTerminalConnection.CISCO_SELECTEDNONE • CiscoTerminalConnection.CISCO_SELECTEDLOCAL • CiscoTerminalConnection.CISCO_SELECTEDREMOTE

Interface	Method	Description
void	startRecording(intplayToneDirection)	Starts recording a call. The system delivers CiscoTermConnRecordingStartEv and CiscoTermConnRecordingTargetInfoEv to the call observer when this method is successful. **Pre-conditions** • ((this.getTerminal()).getProvider()).getState() = = Provider.IN_SERVICE • this.getCallControlState() = = CallControlTerminalConnection.TALKING • ((CiscoProviderCapabilities) (this.getTerminal().getProvider(). getProviderCapabilities()).canRecord() = = TRUE • this.getConnection().getAddress(). getRecordingConfig(this.getTerminal()) = = CiscoAddress. APPLICATION_CONTROLLED__RECORDING **Parameters** • playToneDirection—Specifies whether to play a tone. Valid values are: • CiscoCall. PLAYTONE_NOLOCAL_OR_REMOTE • CiscoCall. PLAYTONE_LOCALONLY • CiscoCall. PLAYTONE_REMOTEONLY • CiscoCall. PLAYTONE_BOTHLOCALANDREMOTE **Throws** • javax.telephony. InvalidStateException—Either the Provider was not "in service" or the TerminalConnection is not in the "TALKING" state. • javax.telephony. PrivilegeViolationException—The application does not have the proper authority to invoke this method. • javax.telephony. ResourceUnavailableException—An internal resource that this method requires is not available. • javax.telephony. InvalidArgumentException—The value for playToneDirection is not valid.
void	startRecording(int playToneDirection, int invocationType)	This method is similar to the startRecording(int playToneDirection) method.

Interface	Method	Description
void	stopRecording(int invocationType)	This method is similar to the stopRecording() method. If an application attempts to stop an active recording, but specifies a recording type other than the recording type that the recording was invoked with, the request fails and an exception with error code CTIERR_RECORDING_INVOCATION _TYPE_NOT_MATCHING is thrown.
CiscoRecorderInfo	getCiscoRecorderInfo()	Returns CiscoRecorderInfo, which exposes the terminal name and address of the recorder or null if the call is not being recorded. The call control terminal connection must be in the talking state.
CiscoMonitor InitiatorInfo	getCiscoMonitorInitiatorInfo()	Returns CiscoMonitorInitiatorInfo or null if the call is not being monitored. The application can use this method on the terminal connection of the monitor target to get information about the monitor initiator or determine that there is no monitor session.
CiscoMonitor TargetInfo	getCiscoMonitorTargetInfo()	Returns CiscoMonitorTargetInfo or null. The application can use this method on the terminal connection of the monitor initiator to get information about the monitor target. This method returns null when called on a terminal connection of the monitor target or if there is no monitor session.
void	addMediaStream(String streamDN, String callingPartyNumber)	Sends a request to begin sending media to the TerminalConnection's associated Built-in-bridge (BIB). **Parameters** • String streamDN—Dialed Number (DN) for the IVR, CTI Port, or the device streaming the media to the call. • String callingPartyNumber—A string object that applications use to provide information to the IVR. This is subject to all the constraints of a DN, and is used to store the agent's DN. This is presented to the IVR as the calling party in the new call event. The IVR must have some application running that understands this information, and can only be retrieved from the new call event on the IVR.
void	removeMediaStream()	Sends a request to cease the playing of media to the TerminalConnection's associated BIB.

Interface	Method	Description
void	playTone(int toneType, int playToneDirection)	Plays tones at local or remote ends of the call. **Parameters** • int tonetype—One of the tones defined in CiscoTone interface. • int playToneDirection—Can be CiscoCall.PLAYTONE_LOCALONLY or CiscoCall.PLAYTONE_REMOTEONLY. **Throws** • InvalidStateException • InvalidArgumentException • PlatformException
void	hold (String contentID)	This interface puts a call on hold, and specifies a contentID that can be used to play video on hold, or other related features. Note that CiscoProvider must be in IN_SERVICE state. Also, the call control state needs to be in TALKING state. If not, InvalidStateException will be thrown. If the hold request fails, ResourceUnavailableException will be thrown. All other errors encountered will result in PlatformException to be thrown. **Parameters** String contentID—A String representing a specific video. Max 128 characters.

Inherited Methods

From Interface javax.telephony.callcontrol.CallControlTerminalConnection

getCallControlState, hold, join, leave, unhold

From Interface javax.telephony.TerminalConnection

answer, getCapabilities, getConnection, getState, getTerminal, getTerminalConnectionCapabilities, getObject, setObject

From Interface com.cisco.jtapi.extensions.CiscoObjectContainer

Related Documentation

See Constant Field Values, on page 1661.

CiscoTerminalObserver

Applications implement this interface to receive CiscoTermEv events such as CiscoRTPInputStartedEv and CiscoRTPInputStoppedEv when observing Terminals via the Terminal.addObserver method.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.TerminalObserver

Declaration

public interface CiscoTerminalObserver extends javax.telephony.TerminalObserver

Fields

None

Methods

None

Inherited Methods

From Interface javax.telephony.TerminalObserver

terminalChangedEvent

Related Documentation

See CiscoTermInServiceEv, CiscoTermOutOfServiceEv, CiscoRTPInputStartedEv, CiscoRTPInputStoppedEv, CiscoRTPOutputStartedEv, and CiscoRTPOutputStoppedEv.

CiscoTerminalProtocol

The CiscoTerminalProtocol event is a container for the constants that define protocol types.

Interface History

Cisco Unified Communications Manager Release	Description
3.x	Added the extension.

Superinterfaces

public interface CiscoTerminalProtocol

Fields

Table 238: Fields in CiscoTerminalProtocol

Interface	Field	Description
static int	PROTOCOL_NONE	This constant value returned by the getProtocol() interface on CiscoTerminal indicates that the protocol type for the CiscoTerminal is not known or not available.
static int	PROTOCOL_SCCP	This constant value returned by the getProtocol() interface on CiscoTerminal indicates that the protocol type for the CiscoTerminal is SCCP.
static int	PROTOCOL_SIP	This constant value returned by the getProtocol() interface on CiscoTerminal indicates that the protocol type for the CiscoTerminal is SIP.
static int	PROTOCOL_CTI_REMOTE_DEVICE	This constant value returned by the getProtocol() interface on CiscoTerminal indicates that the protocol type for the CiscoTerminal is CTI Remote Device.

Related Documentation

See also CiscoTerminal, Constant Field Values, on page 1661 for more information.

CiscoTermInServiceEv

The CiscoTermInServiceEv event gets sent to the application's TerminalObservers to indicate that the CiscoTerminal is ready for operation.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermInServiceEv extends CiscoTermEv

Fields

Table 239: Fields in CiscoTermInServiceEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 240: Methods in CiscoTermInServiceEv

Interface	Method	Description
int	getSupportedEncoding()	Reports whether a terminal is UNICODE-capable by returning the supported encoding. The value of supported encoding may be any of the following constants: • CiscoTerminal.UNKNOWN_ENCODING • CiscoTerminal.ASCII_ENCODING • CiscoTerminal.UCS2UNICODE_ENCODING • CiscoTerminal.NOT_APPLICABLE Returns: An integer value for the supported encoding of this terminal.
int	getLocale()	Returns the locale that this Terminal supports. Returns int values defined in the CiscoLocales interface.
boolean	getDNDStatus()	Returns the current DND status to the application.Returns boolean dndStatus.
int	getDNDOption()	Returns the current DND option to the application. The DND option can be any of the following constants: • CiscoTerminal.DND_OPTION_NONE • CiscoTerminal.DND_OPTION_RINGER_OFF • CiscoTerminal.DND_OPTION_CALL_REJECT Returns int dndOption.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and CiscoLocales

CiscoTermOutOfServiceEv

The CiscoTermOutOfServiceEv event gets sent to the TerminalObservers of an application to indicate that the CiscoTerminal is out-of-service.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoOutOfServiceEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermOutOfServiceEv extends CiscoTermEv, CiscoOutOfServiceEv

Fields

Table 241: Fields in CiscoTermOutOfServiceEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING,

META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN, CAUSE_CALLMANAGER_FAILURE, CAUSE_CTIMANAGER_FAILURE, CAUSE_DEVICE_FAILURE, CAUSE_DEVICE_RESTRICTED, CAUSE_DEVICE_UNREGISTERED, CAUSE_LINE_RESTRICTED, CAUSE_NOCALLMANAGER_AVAILABLE, CAUSE_REHOME_TO_HIGHER_PRIORITY_CM, CAUSE_REHOMING_FAILURE

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface com.cisco.jtapi.extensions.CiscoOutOfServiceEv

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermRegistrationFailedEv

Applications receive this event when TerminalRegistration fails at the provider. The error that getErrorCode() returns explains the problem. On receiving this event, the application should try to reregister the terminal.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
8.5(1)	A new error code, CTI_SECURITY_NOT_ALLOWED, is added.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermRegistrationFailedEv extends CiscoTermEv

Fields

Table 242: Fields in CiscoTermRegistrationFailedEv

Interface	Field	Description
static final int	ID	None
static final int	MEDIA_CAPABILITY_MISMATCH	Registration failed because the Terminal is already registered with a different media capability. Try reregistering with the same capability.
static final int	MEDIA_ALREADY_TERMINATED_NONE	Registration failed because the Terminal is already registered with media termination type none. Try reregistering with Media termination type none.
static final int	MEDIA_ALREADY_TERMINATED_STATIC	Registration failed because the Terminal is already registered with static media termination. Static registration does not allow a second registration. Wait until the terminal unregisters.
static final int	MEDIA_ALREADY_TERMINATED_DYNAMIC	Registration failed because the Terminal is already registered with dynamic media termination. Try reregistering with dynamic media termination.

Interface	Field	Description
static final int	OWNER_NOT_ALIVE	Registration encountered a race condition while attempting to register the Terminal. Try registering the Terminal.
static final int	DB_INITIALIZATION_ERROR	A database initialization error occurred while registering a Terminal. Try registering the Terminal.
static final int	UNKNOWN	Registration failed for an unknown internal reason. Try to reregister the Terminal.
static final int	IP_ADDRESSING_MODE_MISMATCH	Registration failed due to unsupported IP Addressing mode Try to register the Terminal with correct IP Addressing Mode.
public static	CTI_SECURITY_NOT_ALLOWED	The application registers a device in secured mode but eventually is rejected with the error code.

Inherited Fields

From Inteface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Inteface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 243: Methods in CiscoTermRegistrationFailedEv

Interface	Method	Description
int	getErrorCode()	Returns the error code for this exception, as an integer.

Inherited Methods

From Inteface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Inteface javax.telephony.events.TermEv

getTerminal

From Inteface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermRemovedEv

The CiscoTermRemovedEv event gets sent to the provider observer of the application when a CiscoTerminal gets removed from the provider domain.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoTermRemovedEv extends CiscoProvEv

Fields

Table 244: Fields in CiscoTermRemovedEv

Interface	Field
static final int	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 245: Methods in CiscoTermRemovedEv

Interface	Method	Description
javax.telephony.Terminal	getTerminal()	Returns the Terminal that was removed from the provider domain.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermRestrictedEv

Applications see the CiscoTermRestrictedEv event when a user restricts a Terminal from Cisco Unified Communications Manager administration after the application starts running. Applications will not be able to see events for restricted Terminals, or for addresses on those terminals. If a Terminal gets restricted while it is in the in-service state, applications receive this event, and the Terminal and the corresponding addresses move to the out-of-service state.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

Declaration

public interface CiscoTermRestrictedEv extends CiscoProvEv

Fields

Table 246: Fields in CiscoTermRestrictedEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN,

META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 247: Methods in CiscoTermRestrictedEv

Interface	Method	Description
javax.telephony.Terminal	getTerminal	Returns the Terminal that has become restricted.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.ProvEv

getProvider

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661.

CiscoTermSnapshotCompletedEv

If an application comes up after a call is established between two endpoints, for mid-call monitoring the application needs to query Terminal.createSnapshot(). After the call events for all of the Addresses on the Terminal get delivered, the application will get CiscoTermSnapshotCompletedEv. To maintain backward compatibility, these events get generated only when the application enables the snapShotRTPEnabled filter in CiscoTermEvFilter.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermSnapshotCompletedEv extends CiscoTermEv

Fields

Table 248: Fields in CiscoTermSnapshotCompletedEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

None

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoTermEvFilter and Constant Field Values, on page 1661.

CiscoTermSnapshotEv

If an application comes up after a call is established between two endpoints, for mid-call monitoring the application needs to query Terminal.createSnapshot(). The snapshot event, CiscoTermSnapshotEv, gets sent and indicates whether the current media between the endpoints is secure. Applications could also query CiscoMediaCallSecurityIndicator to get the security indicator for a call; however, this event does not contain any KeyMaterial. If there are no calls on any of the lines on the Terminal, applications will only get CiscoTermSnapshotCompletedEv. To maintain backward compatibility, these events get generated only when the application enables the snapShotRTPEnabled filter in CiscoTermEvFilter.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

Declaration

public interface CiscoTermSnapshotEv extends CiscoTermEv

Fields

Table 249: Fields in CiscoTermSnapshotEv

Interface	Field
staticint	ID

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 250: Methods in CiscoTermSnapshotEv

Interface	Method	Description
CiscoMediaCallSecurity Indicator[]	getCiscoMediaCallSecurityIndicator()	Returns the media security status for each active call on this device.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.TermEv

getTerminal

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See CiscoTermEvFilter and Constant Field Values, on page 1661.

CiscoTone

The CiscoTone interface defines CTI Tone constant codes. CiscoToneChangedEv provides the getTone() method to return one of these constants. Only the ZIPZIP tone type is exposed to applications.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.0(1)	Added a new extension.
8.5(1)	Enhanced to include the supported tones that are used as parameters for playTone() method.

Superinterfaces

public interface CiscoTone

Fields

Table 251: Fields in CiscoTone

Interface	Field	Description
Static int	ZIPZIP	This interface defines the integer value for the ZIPZIP tone. The CiscoToneChangedEv.getTone() interface returns an integer value for tone.
public static final int	ZIP	-
public static final int	CALLWAITINGTONE	-

See also CiscoToneChangedEv, Constant Field Values, on page 1661.

CiscoToneChangedEv

The CiscoToneChangedEv event indicates that a tone has been generated for this call. The CallControlCallObserver interface reports this event. Currently, this tone gets generated only because of the Forced Authorization Code (FAC) or Client Matter Code (CMC) features. CiscoToneChangedEv.getCiscoCause() returns CiscoCallEv.CAUSE_FAC_CMC_FEATURE if the tone gets generated because of FAC_CMC.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoToneChangedEv extends CiscoCallEv

Fields

Table 252: Fields in CiscoToneChangedEv

Interface	Field	Description
static final int	ID	None
static final int	FAC_REQUIRED	Indicates that a FAC must be entered using Connection.addToAddress(String) to extend the call. This applies only for FAC_CMC_FEATURE_ID.
static final int	CMC_REQUIRED	Indicates that a CMC must be entered using Connection.addToAddress(String) to extend the call. This applies only for FAC_CMC_FEATURE_ID.
static final int	FAC_CMC_REQUIRED	Indicates that both a FAC and a CMC must be entered using Connection.addToAddress(String) to extend the call. The application can enter either one String code at a time or both at same time, separated by a # (pound sign) character. This applies only for FAC_CMC_FEATURE_ID.

CAUSE_ACCESSINFORMATIONDISCARDED

, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED,
CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST,
CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED,
CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND,
CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION,
CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE,
CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED,
CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE,
CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH,
CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE,
CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK,
CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR,
CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST,
CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL,
CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED,
CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING,
CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED,
CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET,
CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED,
CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,
CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH,
CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL,
CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL,

CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

Inherited Fields

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface com.cisco.jtapi.extensions.CiscoCallEv

Methods

Table 253: Methods in CiscoToneChangedEv

Interface	Method	Description
int	getTone()	Returns the generated tone type from CiscoTone.
int	getWhichCodeRequired()	Returns which codes are required for the dialed DN. The codeRequired may be one of the following: • CiscoToneChangedEv.FAC_REQUIRED • CiscoToneChangedEv.CMC_REQUIRED • CiscoToneChangedEv.FAC_CMC_REQUIRED

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface com.cisco.jtapi.extensions.CiscoCallEv

Related Documentation

See Constant Field Values, on page 1661.

CiscoTransferEndEv

The CiscoTransferEndEv event indicates that a transfer operation has completed. This event gets reported via the CallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoTransferEndEv extends CiscoCallEv

Fields

None

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE,

CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 254: Methods in CiscoTransferEndEv

Interface	Method	Description
javax.telephony.Call	getTransferredCall()	Returns the call that has been transferred. This call is in the Call.INVALID state.
javax.telephony.Call	getFinalCall()	Returns the call that remains active after the transfer completes.
javax.telephony.TerminalConnection	getTransferController()	Returns the ACTIVE TerminalConnection that currently acts as the transfer controller for the final call. When the transferController is a SharedLine, there will be multiple TerminalConnection objects. This method returns the ACTIVE TerminalConnection; however, if the application is not observing the ACTIVE TerminalConnection, this method will return one of the PASSIVE TerminalConnection objects.

Interface	Method	Description
javax.telephony.TerminalConnection[]	getTransferControllers()	Returns the list of TerminalConnection objects that currently act as the transfer controller for the final call. When the transferController is not a SharedLine, there will be only one TerminalConnection in the list. This method returns null if there is no observer on the transfer controller.
javax.telephony.Address	getTransferControllerAddress()	Returns the address that currently acts as the transfer controller for the final call.
boolean	isSuccess()	Returns true if the transfer is successful, or false otherwise.

Inherited Methods

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

Related Documentation

See Constant Field Values, on page 1661 and getTransferControllers().

CiscoTransferStartEv

The CiscoTransferStartEv event indicates that a transfer operation has started. This event gets reported via the CallControlCallObserver interface.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.

Superinterfaces

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

Declaration

public interface CiscoTransferStartEv extends CiscoCallEv

Fields

Table 255: Fields in CiscoTransferStartEv

Interface	Field
static final int	ID

Inherited Fields

From Interface com.cisco.jtapi.extensions.CiscoCallEv

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME, CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED, CAUSE_CTICCMSIP423INTERVALTOOBRIEF, CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE, CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED, CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE, CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE, CAUSE_CTICCMSIP487REQUESTTERMINATED, CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING, CAUSE_CTICCMSIP493UNDECIPHERABLE, CAUSE_CTICCMSIP500SERVERINTERNALERROR, CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY, CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT, CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED, CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,

CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

From Interface javax.telephony.events.Ev

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

Methods

Table 256: Methods in CiscoTransferStartEv

Interface	Method	Description
javax.telephony.Call	getTransferredCall()	Returns the call that will be transferred.
javax.telephony.Call	getFinalCall()	Returns the call that will remain active after the transfer completes.
javax.telephony.TerminalConnection	getTransferController()	Returns the ACTIVE TerminalConnection that currently acts as the transfer controller for the final call. When the transferController is a SharedLine, there will be multiple TerminalConnection objects. This method returns the ACTIVE TerminalConnection; however, if the application is not observing the ACTIVE TerminalConnection, this method will return one of the PASSIVE TerminalConnection objects.
javax.telephony.TerminalConnection[]	getTransferControllers()	Returns a list of TerminalConnection objects that currently act as the transfer controller for the final call. When the transferController is not a SharedLine, there will be only TerminalConnection in the list. This method returns null if there is no observer on the transfer controller.
javax.telephony.Address	getTransferControllerAddress()	Returns the address that currently acts as the transfer controller for the final call.
String	getControllerTerminalName()	Returns the terminal names of the controllers across which transfer is done.

Inherited Methods

From Interface com.cisco.jtapi.extensions.CiscoCallEv

getCiscoCause, getCiscoFeatureReason

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

From Interface javax.telephony.events.CallEv

getCall

From Interface javax.telephony.events.Ev

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

Related Documentation

See Constant Field Values, on page 1661 and getTransferControllers().

CiscoUrlInfo

The CiscoUrlInfo object specifies the properties of the Uniform Resources Locator (URL) associated with a SIP endpoint.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(1 and 2)	Created history table to track changes.
9.0(1)	Two new fields, TRANSPORT_TYPE_UNKNOWN and TRANSPORT_TYPE_TLS, are added.

Declaration

public interface CiscoUrlInfo

Fields

Table 257: Fields in CiscoUrlInfo

Interface	Field	Description
static final int	TRANSPORT_TYPE_UNKNOWN	The endpoint is using an unknown transport type.
static final int	TRANSPORT_TYPE_UDP	The endpoint is using UDP.
static final int	TRANSPORT_TYPE_TCP	The endpoint is using TCP.
static final int	TRANSPORT_TYPE_TLS	The endpoint is using TLS.
static final int	URL_TYPE_UNKNOWN	The URL is of unknown type.
static final int	URL_TYPE_TEL	The URL is of type telephony.
static final int	URL_TYPE_SIP	The URL is of type SIP.

Methods

Table 258: Methods in CiscoUrlInfro

Interface	Method	Description
java.lang.String	getUser()	Returns the user name associated with the SIP endpoint as a string
java.lang.String	getHost()	Returns the host name associated with the SIP endpoint.
int	getPort()	Returns the port associated with the SIP endpoint.
int	getTransportType()	Returns the Transport Layer Protocol type that the SIP endpoint uses. The type is either CiscoUrlInfo.TRANSPORT_TYPE_UDP or CiscoUrlInfo.TRANSPORT_TYPE_TCP.
int	getUrlType()	This method returns the endpoint URL type. CiscoUrlInfo.URL_TYPE_UNKNOWN, CiscoUrlInfo.URL_TYPE_TEL, and CiscoUrlInfo.URL_TYPE_SIP are the possible return values.

Related Documentation

See Constant Field Values, on page 1661.

ComponentUpdater

The overloaded method is introduced which creates an updater log in the directory that is specified.

Interface History

Cisco Unified Communications Manager Release Number	Description
7.1(2)	Added in 7.1(2)

Declaration

public interface ComponentUpdater

Methods

The overloaded method is introduced which creates updater log in the directory specified.

Table 259: Methods in ComponentUpdater

Interface	Method	Description
ComponentUpdater	String tracePath	Returns the Consult Call for which consult operation is cancelled, if the consult call doesn't exist it will return NULL.

Related Documentation

See Constant Field Values, on page 1661.

ProviderPickupNotificationRegistrationClosedEv

ProviderPickupNotificationRegistrationClosedEvent is a new interface being added with Call Pickup feature development. This event is fired whenever something happens on the CUCM that results in a previous registration to observe a pickup group being made invalid. For example, removal of pickup group from the CUCM, or change in Pickup Number etc. Applications should look for these events and handle them accodringly.

Interface History

Cisco Unified Communications Manager Release Number	Description
8.0(1)	New interface

Declaration

public interface ProviderPickupNotificationRegistrationClosedEvent extends CiscoProvEvMethods

Methods

Table 260: Methods in ProviderPickupNotificationRegistrationClosedEv

Interface	Method	Description
String	getPickupGroupNumber()	This method returns the Pickup Group Number for the Pickup Group that is invalidated by the event.
String	getPickupGroupPartition()	This method returns the Pickup Group Partition for the Pickup Group that is invalidated by the event.
int	getReason()	This method returns the reason code explaining why this Pickup Group was invalidated.

New Reason Code

CTIERR_PICKUPGROUP_CHANGED

CTIERR_PICKUPGROUP_DELETED

Related Documentation

None

CiscoTermHuntLogStatusChangedEv

This is a new interface that has been introduced in Cisco JTAPI. The intention of this new interface is to notify the applications with event CiscoTermHuntLogStatusChangedEv whenever the value of hunt log status is changed, provided the filter is set to true on that particular terminal.

Declaration

Methods

Inteface	Method	Description
int	getHuntLogStatus()	This method is used get the value of huntlogstatus of the terminal, it returns either CiscoTerminal.DEVICE_HUNT_LOGGED_IN, CiscoTerminal.DEVICE_HUNT_LOGGED_OUT, or CiscoTerminal. DEVICE_HUNT_NOT_APPLICABLE

CiscoProvConnToLeastPriorCtiServerEv

Interface History

Cisco Unified Communications Manager Release Number	Description
14SU3	New interface.

Declaration

public interface CiscoProvConnToLeastPriorCtiServerEv extends CiscoProvEv.

Fields

Table 261: Fields in CiscoProvConnToLeastPriorCtiServerEv

Interface	Field	Description
Static int	ID	

Methods

Related Documentation

None

CiscoProvFallbackToPrimNwCompltdEv

Interface History

Cisco Unified Communications Manager Release Number	Description
14SU3	New interface.

Declaration

public interface CiscoProvFallbackToPrimNwCompltdEv extends CiscoProvEv.

Fields

Table 262: Fields in CiscoProvFallbackToPrimNwCompltdEv

Interface	Field	Description
Static int	ID	

Methods

Related Documentation

None

CiscoProvPrimNwReachableEv

Interface History

Cisco Unified Communications Manager Release Number	Description
14SU3	New interface.

Declaration

public interface CiscoProvPrimNwReachableEv extends CiscoProvEv.

Fields

None

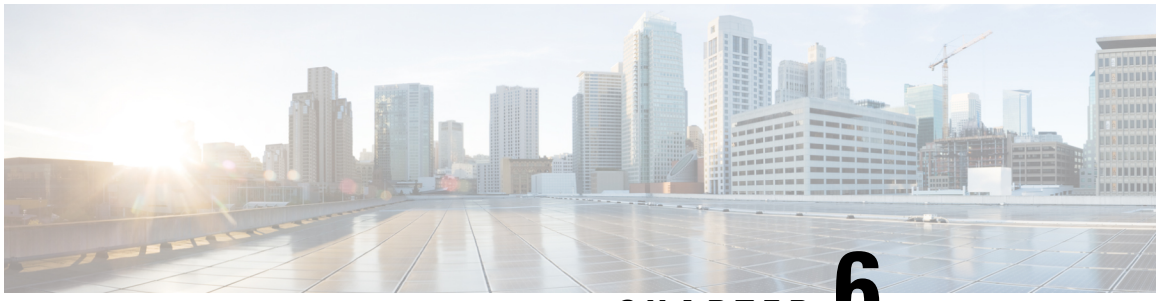
Methods

Table 263: Methods in CiscoProvPrimNwReachableEv

Interface	Method	Description
String[]	getReachableCtiServers()	Returns a list of CTI Servers that are reachable after application fails over to the least priority CTI Server.

Related Documentation

None



CHAPTER 6

Cisco Unified JTAPI Alarms and Services

The Cisco Unified JTAPI alarms and services consists of a set of classes and interfaces that expose the additional functionality not readily exposed in JTAPI 1.2 specification but are available in Cisco Unified Communications Manager. Developers can use the classes and interfaces to create new applications or modify existing classes and interfaces to create new methods.

This chapter describes the alarms and services that are available for implementation in a Cisco Unified Communications Manager.

For information about Cisco Unified JTAPI extensions, see Cisco Unified JTAPI Extensions, on page 243

Alarm Class Hierarchy

The following class hierarchy is contained in the com.cisco.services.alarm package.

```
java.lang.Object
   com.cisco.services.alarm.AlarmManager, on page 670
com.cisco.services.alarm.DefaultAlarm, on page 674 (implements
                                                    com.cisco.services.alarm.Alarm)
com.cisco.services.alarm.DefaultAlarmWriter, on page 676 (implements
                           com.cisco.services.alarm.AlarmWriter, on page 687)
com.cisco.services.alarm.ParameterList, on page 680
```

AlarmManager

The AlarmManager is used to create Alarm objects. The AlarmManager is created with a facility and AlarmService hostname and port. All alarms created by the factory will be associated with this facility. This class also maintains a reference to a single AlarmWriter that can be used system wide. An application can make use of this AlarmWriter. AlarmManager exposes a default implementation of an AlarmWriter. Applications can override this with a user defined implementation of their own AlarmWriter.

Usage

AlarmManager AlarmManager = new AlarmManager(facilityName, alarmServiceHost, alarmServicePort, debugTrace, errorTrace);

Alarms are created by the factory by supplying the alarmName (mnemonic), subfacility and severity Alarms can be cached for use in different parts of the application. During a send alarm applications can specify the variable parameters that offer specific information to the AlarmService.

Usage

Typically applications will maintain their own AlarmManager instance. Applications will also have to set a debug and error trace to enable the alarm tracing to also be sent to the existing trace destinations.

Setup the manager and writer classes:

AlarmWriter alarmWriter = new DefaultAlarmWriter(port, alarmServiceHost);

AlarmManager alarmManager = new AlarmManager(“AA_IVR”, alarmWriter, debugTrace, errorTrace);

Generating the Alarms:

create an alarm for the subfacility and a default severity.

Alarm alarm = alarmManager.createAlarm(“HTTPSS”, Alarm.INFORMATIONAL);

alarm.send(“090T”) sends the alarm with the mnemonic

alarm.send(“090T”, “Port is stuck”, “CTIPort01”) or with a mnemonic and parameter

Declaration

```
public class AlarmManager

java.lang.Object
 |
 +--com.cisco.services.alarm.AlarmManager
```

Note More than one parameter can be sent by specifying a ParameterList

Member summary	
Constructors	
	AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace), on page 671 Create an instance of the AlarmManager for the facility.
Methods	
`Alarm`	createAlarm(String, int), on page 672 Creates an Alarm of required severity for the subFacility
`AlarmWriter`	getAlarmWriter(), on page 672
`void`	setAlarmWriter(AlarmWriter), on page 672 Allows applications to override the AlarmWriter to be used by this AlarmManager, with a user defined AlarmWriter

Inherited member summary
Methods inherited from class `Object`
`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()`

Constructors

AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace)

```
public AlarmManager(java.lang.String facility,
  com.cisco.services.alarm.AlarmWriterwriter,
  com.cisco.services.tracing.TracedebugTrace_,
  com.cisco.services.tracing.UnconditionalTraceerrorTrace_)
```

Create an instance of the AlarmManager for the facility. Applications specify an AlarmWriter to be used by this AlarmManager to send the Alarms to the AlarmService.

Methods

createAlarm(String, int)

```
public com.cisco.services.alarm.Alarm createAlarm
   (java.lang.String subfacility,  intseverity)
```

Creates an Alarm of required severity for the subFacility

Returns:

an object implementing the alarm interface

getAlarmWriter()

```
public com.cisco.services.alarm.AlarmWriter getAlarmWriter()
```

Returns:

an AlarmWriter object

setAlarmWriter(AlarmWriter)

```
public void setAlarmWriter(com.cisco.services.alarm.AlarmWriter writer)
```

Allows applications to override the AlarmWriter to be used by this AlarmManager, with a user defined AlarmWriter

AlarmWriter

An AlarmWriter receives alarm messages and transmits it to the receiving AlarmService on a TCP link. This interface can be used to implement other AlarmWriters to be used with this implementation of com.cisco.service.alarm A DefaultAlarmWriter is provided with this implementation and can be obtained from the AlarmManager.

Declaration

```
public interface AlarmWriter
```

All Known Implementing Classes

DefaultAlarmWriter, on page 676

Member Summary

Member summary	
Methods	
`void`	close(), on page 673 close the AlarmWriter

Member summary	
`java.lang.String`	getDescription(), on page 673
`boolean`	getEnabled(), on page 673
`java.lang.String`	getName(), on page 673
`void`	send(String), on page 673 Send out the alarm message to the AlarmService.
`void`	setEnabled(boolean), on page 674 Enable or disable the AlarmWriter

Methods

close()

```
public void close()
```

close the AlarmWriter

getDescription()

```
public java.lang.String getDescription()
```

Returns:

the AlarmWriter description

getEnabled()

```
public boolean getEnabled()
```

Returns:

the current enabled or disabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Returns:

the AlarmWriter name

send(String)

```
public void send(java.lang.String alarmMessage)
```

Send out the alarm message to the AlarmService.

Parameters:

the - Alarm to be sent

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Enable or disable the AlarmWriter

Parameters:

enable or disable the AlarmWriter

DefaultAlarm

An Implementation of the Alarm interface. The AlarmManager creates these Alarms when the createAlarm() method is called.

Declaration

```
public class DefaultAlarm implements Alarm, on page 682

java.lang.Object
|
+--com.cisco.services.alarm.DefaultAlarm
```

All Implemented Interfaces

Alarm, on page 682

Member Summary

Member summary	
Constructors	
	DefaultAlarm(String, String, int, AlarmWriter), on page 675
Methods	
`java.lang.String`	getFacility(), on page 675
`int`	getSeverity(), on page 675
`java.lang.String`	getSubFacility(), on page 675
`void`	send(String), on page 675 Send the alarm with the specified mnemonic
`void`	send(String, ParameterList), on page 676 Send the alarm with the specified name and list of parameters.
`void`	send(String, String, String), on page 676 Send the alarm with the specified name and parameter

Inherited member summary
Fields inherited from interface Alarm, on page 682
ALERTS, on page 685, CRITICAL, on page 685, DEBUGGING, on page 685, EMERGENCIES, on page 685, ERROR, on page 685, HIGHEST_LEVEL, on page 685, INFORMATIONAL, on page 686, LOWEST_LEVEL, on page 686, NOTIFICATION, on page 686, NO_SEVERITY, on page 686, UNKNOWN_MNEMONIC, on page 686, WARNING, on page 686
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

DefaultAlarm(String, String, int, AlarmWriter)

```
public DefaultAlarm(java.lang.String facility,
   java.lang.StringsubFacility,
   intseverity,
   com.cisco.services.alarm.AlarmWriteralarmWriter)
```

Methods

getFacility()

```
public java.lang.String getFacility()
```

Specified By:

getFacility(), on page 686 in interface Alarm, on page 682

getSeverity()

```
public int getSeverity()
```

Specified By:

getSeverity(), on page 686 in interface Alarm, on page 682

getSubFacility()

```
public java.lang.String getSubFacility()
```

Specified By:

getSubFacility(), on page 687 in interface Alarm, on page 682

send(String)

```
public void send(java.lang.String mnemonic)
```

Send the alarm with the specified mnemonic

Specified By:

send(String), on page 687 in interface Alarm, on page 682

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
   com.cisco.services.alarm.ParameterListparamList)
```

Send the alarm with the specified name and list of parameters.

Specified By:

send(String, ParameterList), on page 687 in interface Alarm, on page 682

send(String, String, String)

```
public void send(java.lang.String mnemonic,
   java.lang.StringparamName,
   java.lang.StringparamValue)
```

Send the alarm with the specified name and parameter

Specified By:

send(String, String, String), on page 687 in interface Alarm, on page 682

DefaultAlarmWriter

`DefaultAlarmWriter` implementation of the AlarmWriter interface.

`DefaultAlarmWriter` maintains a queue of a fixed size to which the alarms are written. The sending of the alarms to the alarm service takes place on a separate thread. The queue is fixed size.

Declaration

```
public class DefaultAlarmWriter implements AlarmWriter, on page 672

java.lang.Object
|
+--com.cisco.services.alarm.DefaultAlarmWriter
```

All Implemented Interfaces

AlarmWriter, on page 672

Member Summary

Member summary	
Constructors	
	DefaultAlarmWriter(int, String), on page 677 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and a queue size of fifty (50).

Member summary	
	DefaultAlarmWriter(int, String, int), on page 678 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size.
	DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace), on page 678 Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size.
Methods	
`void`	close(), on page 678 Shutdown the send thread and close the socket
`java.lang.String`	getDescription(), on page 679
`boolean`	getEnabled(), on page 679
`java.lang.String`	getName(), on page 679
`static void`	main(String[]), on page 679
`void`	send(String), on page 679 send the Alarm to the alarm service
`void`	setEnabled(boolean), on page 679 Applications can dynamically enable or disable the AlarmWriter

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Constructors

DefaultAlarmWriter(int, String)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and a queue size of fifty (50). The AlarmService is listening on this port for Alarm messages.

Parameters:

port: port on which the alarm service is listening

alarmServiceName: The host name of the machine with the Alarm service

Throws:

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName,
   intqueueSize) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size. The AlarmService is listening on this port for Alarm messages.

Parameters:

port—port on which the alarm service is listening

alarmServiceName—The host name of the machine with the Alarm service

queueSize - the size of the queue to be maintained in the alarm writer

Throws:

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace)

```
public DefaultAlarmWriter(int port,
   java.lang.StringalarmServiceName,
   intqueueSize,
   com.cisco.services.tracing.ConditionalTracedebugTrace_,
   com.cisco.services.tracing.UnconditionalTraceerrorTrace_) throwsUnknownHostException
```

Constructor for the DefaultAlarmWriter which takes the AlarmService hostname, port and queue size. The AlarmService is listening on this port for Alarm messages.

Parameters:

port—port on which the alarm service is listening

alarmServiceName—The host name of the machine with the Alarm service

queueSize - the size of the queue to be maintained in the alarm writer

Throws:

java.net.UnknownHostException

Methods

close()

```
public void close()
```

Shutdown the send thread and close the socket

Specified By:

close in interface AlarmWriter, on page 672

getDescription()

```
public java.lang.String getDescription()
```

Specified By:

getDescription in interface AlarmWriter, on page 672

Returns:

a short description of the AlarmWriter

getEnabled()

```
public boolean getEnabled()
```

Specified By:

getEnabled in interface AlarmWriter, on page 672

Returns:

the enabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Specified By:

getName in interface AlarmWriter, on page 672

Returns:

the name of the AlarmWriter

main(String[])

```
public static void main(java.lang.String[] args)
```

send(String)

```
public void send(java.lang.String alarmMessage)
```

send the Alarm to the alarm service

Specified By:

send in interface AlarmWriter, on page 672

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Applications can dynamically enable or disable the AlarmWriter

Specified By:

setEnabled in interface AlarmWriter, on page 672

ParameterList

ParameterList is a list of name value pairs that is used to send additional (and optional) user defined parameters to the AlarmService. These parameters can contain the specifics of an Alarm.

As an example, a LowResourceAlarm can have a parameter that informs the service which particular resource is low:

name = “CPUUsage”

value = “0.9”

These parameters are user definable but must, however, also be pre-defined in the AlarmService catalog.

Declaration

```
public class ParameterList

java.lang.Object
|
+--com.cisco.services.alarm.ParameterList
```

Member Summary

Member summary	
Constructors	
	ParameterList(), on page 681 Default constructor for the ParameterList
	ParameterList(String, String), on page 681 Constructor that takes a name value pair.
Methods	
`void` addParameter(String, String), on page 681	addParameter(String, String), on page 681 method used to add additional name value pairs (parameters) to the list
`java.lang.String[]`	getParameterNames(), on page 681 Get the parameter names in the list
`java.lang.String`	getParameterValue(String), on page 681 get the value for a parameter
`void`	removeAllParameters(), on page 682 remove all the parameters in the list

Member summary	
`void`	removeParameter(String), on page 682 remove a particular parameter if it is in the list
`java.lang.String`	toString(), on page 682

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

ParameterList()

```
public ParameterList()
```

Default constructor for the ParameterList

ParameterList(String, String)

```
public ParameterList(java.lang.String name,
   java.lang.Stringvalue)
```

Constructor that takes a name value pair.

Methods

addParameter(String, String)

```
public void addParameter(java.lang.String name,
   java.lang.Stringvalue)
```

method used to add additional name value pairs (parameters) to the list

getParameterNames()

```
public java.lang.String[] getParameterNames()
```

Get the parameter names in the list

Returns:

array of parameters

getParameterValue(String)

```
public java.lang.String getParameterValue(java.lang.String parameterName)
```

get the value for a parameter

Returns:

value of a parameter

removeAllParameters()

```
public void removeAllParameters()
```

remove all the parameters in the list

removeParameter(String)

```
public void removeParameter(java.lang.String parameterName)
```

remove a particular parameter if it is in the list

toString()

```
public java.lang.String toString()
```

Overrides:

`toString` in class `Object`

Alarm Interface Hierarchy

The following interface hierarchy is contained in the com.cisco.services.alarm package.

com.cisco.services.alarm.Alarm, on page 682

com.cisco.services.alarm.AlarmWriter, on page 687

Alarm

The Alarm interface is used to define Alarms in. An Alarm has an XML representation that it must adhere to in order to be recognized by the Alarm Service, with a DTD as shown below. An application can implement this interface or use the AlarmFactory to generate Alarms of the correct format. The Alarm is the a specification that needs to be sent to an AlarmService that will take some action based on the Alarm. Using this specification the AlarmService will access definitions available in a catalog. This catalog is maintained by the user requiring the Alarm function to effect the appropriate action for the Alarm. The severity specified the Alarm can over-ride the severity associated with this Alarm in the catalog. If no severity is specified in the Alarm the catalog severity is used.

Alarm severities are derived from Syslog and are defined as follows:

0 = EMERGENCIES System unusable

1 = ALERTS Immediate action needed

2 = CRITICAL Critical conditions

3 = ERROR Error conditions

4 = WARNING Warning conditions

5 = NOTIFICATION Normal but significant condition

6 = INFORMATIONAL Informational messages only

7 = DEBUGGING Debugging messages

Declaration

```
public interface Alarm
```

All Known Implementing Classes

DefaultAlarm, on page 674

Member Summary

Member summary	
Fields	
`static int`	ALERTS, on page 685 The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed) Syslog severity level = 1
`static int`	CRITICAL, on page 685 A critical failure, the application cannot accomplish the tasks required due to this failure, for example, the application cannot open the database to read the device list Syslog severity level = 2
`static int`	DEBUGGING, on page 685 Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled Syslog severity level = 7
`static int`	EMERGENCIES, on page 685 Emergency situation, a system shutdown is necessary Syslog severity level = 0
`static int`	ERROR, on page 685 An error condition of some kind has occurred and the user needs to understand the nature of that failure Syslog severity level = 3

Member summary	
`static int`	HIGHEST_LEVEL, on page 685 The highest trace level, currently this is DEBUGGING with a trace level of 7
`static int`	INFORMATIONAL, on page 686 Information of some form not relating to errors, warnings, audit, or debug Syslog severity level = 6
`static int`	LOWEST_LEVEL, on page 686 The lowest trace level, currently this is EMERGENCIES with a trace level of 0
`static int`	NO_SEVERITY, on page 686 Applications can set this level to generate Alarms without a severity.
`static int`	NOTIFICATION, on page 686 Notification denotes a normal but significant condition Syslog severity level = 5
`static java.lang.String`	UNKNOWN_MNEMONIC, on page 686 String used when a mnemonic is not specified during an Alarm send
`static int`	WARNING, on page 686 Warning that a problem of some form exists but is not keeping the application from completing its tasks Syslog severity level = 4
Methods	
`java.lang.String`	getFacility(), on page 686
`int`	getSeverity(), on page 686
`java.lang.String`	getSubFacility(), on page 687
`void`	send(String), on page 687 send the Alarm with the specified mnemonic.
`void`	send(String, ParameterList), on page 687 send an Alarm with the specified mnemonic and supplied parameter list

Member summary	
`void`	send(String, String, String), on page 687 send an Alarm with the specified mnemonic and with one parameter

Fields

ALERTS

```
public static final int ALERTS
```

The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed)

Syslog severity level = 1

CRITICAL

```
public static final int CRITICAL
```

A critical failure, the application cannot accomplish the tasks required due to this failure, for example, the application cannot open the database to read the device list

Syslog severity level = 2

DEBUGGING

```
public static final int DEBUGGING
```

Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled (Syslog severity level = 7).

EMERGENCIES

```
public static final int EMERGENCIES
```

Emergency situation, a system shutdown is necessary

Syslog severity level = 0

ERROR

```
public static final int ERROR
```

An error condition of some kind has occurred and the user needs to understand the nature of that failure

Syslog severity level = 3

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

The highest trace level, currently this is DEBUGGING with a trace level of 7

INFORMATIONAL

```
public static final int INFORMATIONAL
```

Information of some form not relating to errors, warnings, audit, or debug

Syslog severity level = 6

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

The lowest trace level, currently this is EMERGENCIES with a trace level of 0

NO_SEVERITY

```
public static final int NO_SEVERITY
```

Applications can set this level to generate Alarms without a severity. NOTE: This is only intended for cases where an application wants the AlarmService to use the severity associated with the Alarm in the catalog

NOTIFICATION

```
public static final int NOTIFICATION
```

Notification denotes a normal but significant condition (Syslog severity level = 5).

UNKNOWN_MNEMONIC

```
public static final java.lang.String UNKNOWN_MNEMONIC
```

String used when a mnemonic is not specified during an Alarm send

WARNING

```
public static final int WARNING
```

Warning that a problem of some form exists but is not keeping the application from completing its tasks (Syslog severity level = 4).

Methods

getFacility()

```
public java.lang.String getFacility()
```

Returns:

the facility name of this Alarm

getSeverity()

```
public int getSeverity()
```

Returns:

severity of the alarm, an integer in the range [0-7]

getSubFacility()

```
public java.lang.String getSubFacility()
```

Returns:

the subfacility of this Alarm

send(String)

```
public void send(java.lang.String mnemonic)
```

send the Alarm with the specified mnemonic. If a null or empty String is passed a mnemonic UNK is sent

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
   com.cisco.services.alarm.ParameterListparameterList)
```

send an Alarm with the specified mnemonic and supplied parameter list

send(String, String, String)

```
public void send(java.lang.String mnemonic,
   java.lang.StringparameterName,
   java.lang.StringparameterValue)
```

send an Alarm with the specified mnemonic and with one parameter.

AlarmWriter

An AlarmWriter receives alarm messages and transmits it to the receiving AlarmService on a TCP link. This interface can be used to implement other AlarmWriters to be used with this implementation of com.cisco.service.alarm A DefaultAlarmWriter is provided with this implementation and can be obtained from the AlarmManager.

Declaration

```
public interface AlarmWriter
```

All Known Implementing Classes

DefaultAlarmWriter, on page 676

Member Summary

Member summary	
Methods	
void	close(), on page 673 close the AlarmWriter

Member summary	
`java.lang.String`	getDescription(), on page 673
`boolean`	getEnabled(), on page 673
`java.lang.String`	getName(), on page 673
`void`	send(String), on page 673 Send out the alarm message to the AlarmService.
`void`	setEnabled(boolean), on page 674 Enable or disable the AlarmWriter

Methods

close()

```
public void close()
```

close the AlarmWriter

getDescription()

```
public java.lang.String getDescription()
```

Returns:

the AlarmWriter description

getEnabled()

```
public boolean getEnabled()
```

Returns:

the current enabled or disabled state of the AlarmWriter

getName()

```
public java.lang.String getName()
```

Returns:

the AlarmWriter name

send(String)

```
public void send(java.lang.String alarmMessage)
```

Send out the alarm message to the AlarmService.

Parameters:

the Alarm to be sent

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

Enable or disable the AlarmWriter

Parameters:

enable or disable the AlarmWriter

Services Tracing Class Hierarchy

The following class hierarchy is contained in the com.cisco.services.tracing package.

```
java.lang.Object
   com.cisco.services.tracing.BaseTraceWriter, on page 689 (implements
                                                com.cisco.services.tracing.TraceWriter)
   com.cisco.services.tracing.ConsoleTraceWriter, on page 693
   com.cisco.services.tracing.LogFileTraceWriter, on page 695
    com.cisco.services.tracing.OutputStreamTraceWriter, on page 701
   com.cisco.services.tracing.SyslogTraceWriter, on page 704
com.cisco.services.tracing.TraceManagerFactory, on page 706
```

BaseTraceWriter

This abstract class is useful for supplying a default, non-printing TraceWriter to a TraceWriterManager This class must be extended to provide the functionality to trace to different streams. The doPrintln() method must be implemented by the extending class.

Declaration

```
public abstract class BaseTraceWriter implements TraceWriter, on page 722

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 722

Direct Known Subclasses

ConsoleTraceWriter, on page 693, LogFileTraceWriter, on page 695, OutputStreamTraceWriter, on page 701, SyslogTraceWriter, on page 704

Member Summary

Member summary	
Constructors	
`protected`	BaseTraceWriter(int[], String, String), on page 691 BaseTraceWriter with trace levels as passed in traceLevels in the array falling outside the range Trace.LOWEST_LEVEL and Trace.HIGHEST_LEVEl are ignored
`protected`	BaseTraceWriter(int, String, String), on page 691 BaseTraceWriter that traces all levels up to the maxTraceLevel The trace level is maintained in the range [Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL]
`protected`	BaseTraceWriter(String, String), on page 691 BaseTraceWriter which only traces the lowest level i.e. severity level, Trace.LOWEST_LEVEL messages
Methods	
`void`	close(), on page 691
`protected void`	doClose(), on page 692
`protected void`	doFlush(), on page 692
`protected abstract void`	doPrintln(String, int), on page 692 Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality
`void`	flush(), on page 692
`java.lang.String`	getDescription(), on page 692
`boolean`	getEnabled(), on page 692
`java.lang.String`	getName(), on page 692
`int[]`	getTraceLevels(), on page 692
`void`	println(String, int), on page 693
`void`	setTraceLevels(int[]), on page 693
`java.lang.String`	toString(), on page 693

Inherited member summary
Methods inherited from class `Object`
clone() , equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

BaseTraceWriter(int[], String, String)

```
protected BaseTraceWriter(int[] traceLevels,
   java.lang.Stringname,
   java.lang.Stringdescription)
```

BaseTraceWriter with trace levels as passed in traceLevels in the array falling outside the range Trace.LOWEST_LEVEL and Trace.HIGHEST_LEVEl are ignored

Parameters:

traceLevels - array of trace levels

See Also:

Trace, on page 708

BaseTraceWriter(int, String, String)

```
protected BaseTraceWriter(int maxTraceLevel,
   java.lang.Stringname,
   java.lang.Stringdescription)
```

BaseTraceWriter that traces all levels up to the maxTraceLevel The trace level is maintained in the range [Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL]

See Also:

Trace, on page 708

BaseTraceWriter(String, String)

```
protected BaseTraceWriter(java.lang.String name,
   java.lang.Stringdescription)
```

BaseTraceWriter which only traces the lowest level i.e. severity level, Trace.LOWEST_LEVEL messages

See Also:

Trace, on page 708

Methods

close()

```
public final void close()
```

Description copied from interface:

com.cisco.services.tracing.TraceWriter

Releases any resources associated by this TraceWriter.

Specified By:

`close` in interface TraceWriter, on page 722

doClose()

```
protected void doClose()
```

doFlush()

```
protected void doFlush()
```

doPrintln(String, int)

```
protected abstract void doPrintln(java.lang.String message,
   intmessageNumber)
```

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

flush()

```
public final void flush()
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

Forces output of any messages that have been printed using the `println` method

Specified By:

`flush` in interface TraceWriter, on page 722

getDescription()

```
public final java.lang.String getDescription()
```

Specified By:

`getDescription` in interface TraceWriter, on page 722

getEnabled()

```
public boolean getEnabled()
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

Returns whether the println method will print anything or not. A closed TraceWriter will always return `false` from this method.

Specified By:

`getEnabled` in interface TraceWriter, on page 722

getName()

```
public final java.lang.String getName()
```

Specified By:

`getName` in interface TraceWriter, on page 722

getTraceLevels()

```
public final int[] getTraceLevels()
```

Specified By:

`getTraceLevels` in interface TraceWriter, on page 722

println(String, int)

```
public final void println(java.lang.String message,
   intseverity)
```

Description copied from interface: `com.cisco.services.tracing.TraceWriter`

Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream. Each trace writer has a notion of the highest level trace it traces.

Specified By:

`println` in interface TraceWriter, on page 722

setTraceLevels(int[])

```
public final void setTraceLevels(int[] levels)
```

Description copied from interface: com.cisco.services.tracing.TraceWriter

set the trace levels that will be traced by this TraceWriter

Specified By:

`setTraceLevels` in interface TraceWriter, on page 722

toString()

```
public final java.lang.String toString()
```

Overrides:

`toString` in class `Object`

ConsoleTraceWriter

Supplies a console TraceWriter to trace to System.out.

See Also:

Trace, on page 708

Declaration

```
public final class ConsoleTraceWriter extends BaseTraceWriter

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.ConsoleTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 722

Member Summary

Member summary	
Constructors	
	ConsoleTraceWriter(), on page 694 Default constructor, traces all severity levels
	ConsoleTraceWriter(int), on page 695 Constructor that sets the maximum level to be traced.
	ConsoleTraceWriter(int[]), on page 695 Construct a ConsoleTraceWriter with an array of trace levels Only traces with the severity in the tracelevel array are traced
Methods	
`protected void`	doFlush(), on page 695
`protected void`	doPrintln(String, int), on page 695
`static void`	main(String[]), on page 695

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 689
close(), on page 691, doClose(), on page 692, flush(), on page 692, getDescription(), on page 692, getEnabled(), on page 692, getName(), on page 692, getTraceLevels(), on page 692, println(String, int), on page 693, setTraceLevels(int[]), on page 693, toString(), on page 693
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

ConsoleTraceWriter()

```
public ConsoleTraceWriter()
```

Default constructor, traces all severity levels

ConsoleTraceWriter(int)

```
public ConsoleTraceWriter(int maxTraceLevel)
```

Constructor that sets the maximum level to be traced.

See Also:

Trace, on page 708

ConsoleTraceWriter(int[])

```
public ConsoleTraceWriter(int[] traceLevels)
```

Construct a ConsoleTraceWriter with an array of trace levels Only traces with the severity in the tracelevel array are traced

Parameters:

`int` - [] traceLevels

See Also:

Trace, on page 708

Methods

doFlush()

```
protected final void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 689

doPrintln(String, int)

```
protected final void doPrintln(java.lang.String message,
   intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 689

main(String[])

```
public static void main(java.lang.String[] args)
```

LogFileTraceWriter

This class extends the BaseTraceWriter class to implement a TraceWriter that writes to a set of log files, rotating among them as each becomes filled to a specified capacity and stores them in a specified directory.

Each of the log files is named according to a pattern controlled by three properties, CurrentFile, FileNameBase, and FileExtension. The CurrentFile property determines which log file, by ordinal number, is being written at present, the FileNameBase property determines the prefix of each log file name, and the FileExtension property determines the suffix, e.g. “txt”. From these properties, log files are named **FileNameBase LeadingZeroPadding CurrentFile.FileExtension**. The CurrentFile property takes on a value from 1 to the value of the MaxFiles property. Note that the CurrentFile property, when converted to a String, is padded with leading zeroes depending on the values of the MaxFiles and CurrentFile properties. An index file tracks the index of the last file written. If the logFileWriter is recreated (for example if an application is restarted) new files will continue from the last written index.

Where the log files are stored is determined by the path, dirNameBase, useSameDir. If a path is not specified, the current path is used as default. If a dirNameBase is not specified, it write log files in the path. Depending upon whether useSameDir is true or false, files are written to the same directory or a new directory, each time an instance of LogFileTraceWriter is created. In case new directories are being made each time, the directory name will consist of the dirNameBase and a number, separated by an ’_’. The number is one more than the greatest number associated with directories with the same dirNameBame in the path. While specifying the path, you may use either a “/” or “\\”, but not “\”

The LogFileTraceWriter keeps track of how many bytes have been written to the current log file. When that number grows within approximately LogFileTraceWriter.ROLLOVER_THRESHOLD bytes, tracing continues to the next file, which is either CurrentFile + 1 if CurrentFile is not equal to MaxFiles, or 1 if CurrentFile is equal to MaxFiles.

Note All properties of this class are specified in the constructor; there is no way to change them dynamically. **Caveat:** If two instances of LogFileTraceWriter are created with the same path and dirNameBase, and useSameDir is true, they may write to the same file.

Example

The following code instantiates a LogFileTraceWriter that will create log files called “MyLog01.log” through “MyLog12.log”. Each file will grow to approximately 100K bytes in size before the next file is created:

LogFileTraceWriter out = new LogFileTraceWriter (“MyLog”, “log”, 12, 100 * 1024); will create a log file TraceWriter which will rotate traces to 12 files from Mylog01.log and Mylog12.log with a file size of 100 KBytes. By default the tracing is set to the HIGHEST_LEVEL.

The following code constructs a LogFileTraceWriter which stores the log files in the path “c:/LogFiles” in a sub directory, “Run”. The files will be named MyLogXX.log. The number of rotating files will be 12 with a size of 100 KB. The same directory gets used for each instance of the application.

LogFileTraceWriter out = new LogFileTraceWriter (“c:/logFiles”, “Run”, “MyLog”, “log”, 12, 100*1024, true);

See Also

Trace, on page 708

Declaration

```
public final class LogFileTraceWriter extends BaseTraceWriter, on page 689

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.LogFileTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 722

Member Summary

Member summary	
Fields	
`static java.lang.String`	DEFAULT_FILE_NAME_BASE, on page 698
`static java.lang.String`	DEFAULT_FILE_NAME_EXTENSION, on page 699
`static char`	DIR_BASE_NAME_NUM_SEPERATOR, on page 699
`static int`	MIN_FILE_SIZE, on page 699
`static int`	MIN_FILES, on page 699
`static int`	ROLLOVER_THRESHOLD, on page 699
Constructors	
	LogFileTraceWriter(String, String, int, int), on page 699 Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.
	LogFileTraceWriter(String, String, String, String, int, int, boolean), on page 699 Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.
	LogFileTraceWriter(String, String, String, String, int, int, int, boolean), on page 699 Constructs a LogFileTraceWriter that rotates among an arbitrary number of files storing them in a specified directory.
Methods	
`protected void`	doClose(), on page 700 Closes this OutputStream.

Member summary	
`protected void`	doFlush(), on page 700
`protected void`	doPrintln(String, int), on page 700
`int`	getCurrentFile(), on page 700 Returns the CurrentFile property
`java.lang.String`	getFileExtension(), on page 700 Returns the FileExtension property
`java.lang.String`	getFileNameBase(), on page 701 Returns the FileNameBase property
`java.lang.String`	getHeader(), on page 701 Get the header string that will be written at the beginning of each log file.
`int`	getMaxFiles(), on page 701 Returns the MaxFiles property
`int`	getMaxFileSize(), on page 701 Returns the MaxFileSize property
`void`	setHeader(String), on page 701 Set the constant header string that will be written at the beginning of every file, trace writing continues from the next line after the header is written.

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 689
close(), on page 691, doClose(), on page 692, flush(), on page 692, getDescription(), on page 692, getEnabled(), on page 692, getName(), on page 692, getTraceLevels(), on page 692, println(String, int), on page 693, setTraceLevels(int[]), on page 693, toString(), on page 693
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Fields

DEFAULT_FILE_NAME_BASE

```
public static final java.lang.String DEFAULT_FILE_NAME_BASE
```

DEFAULT_FILE_NAME_EXTENSION

```
public static final java.lang.String DEFAULT_FILE_NAME_EXTENSION
```

DIR_BASE_NAME_NUM_SEPERATOR

```
public static final char DIR_BASE_NAME_NUM_SEPERATOR
```

MIN_FILE_SIZE

```
public static final int MIN_FILE_SIZE
```

MIN_FILES

```
public static final int MIN_FILES
```

ROLLOVER_THRESHOLD

```
public static final int ROLLOVER_THRESHOLD
```

Constructors

LogFileTraceWriter(String, String, int, int)

```
public LogFileTraceWriter(java.lang.String fileNameBase,
   java.lang.StringfileNameExtension,
   intmaxFiles,
   intmaxFileSize) throwsIOException
```

Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels. Since a path and Directory Base name is not specified, it writes the files to the current directory without any sub directories.

Throws:

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
   java.lang.StringdirNameBase,
   java.lang.StringfileNameBase,
   java.lang.StringfileNameExtension,
   intmaxFiles,
   intmaxFileSize,
   booleanuseSameDir) throwsIOException
```

Default constructor for LogFileTraceWriter that rotates among an arbitrary number of files with tracing for all levels.

Throws:

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
   java.lang.StringdirNameBase,
   java.lang.StringfileNameBase,
```

```
java.lang.StringfileNameExtension,
intmaxFiles,
intmaxFileSize,
intmaxTraceLevel,
booleanuseSameDir) throwsIOException
```

Constructs a LogFileTraceWriter that rotates among an arbitrary number of files storing them in a specified directory.

Throws:

java.io.IOException

Methods

doClose()

```
protected void doClose()
```

Closes this OutputStream. Any log file that is currently open will be closed as well.

Overrides:

`doClose` in class BaseTraceWriter, on page 689

doFlush()

```
protected void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 689

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
   intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 689

getCurrentFile()

```
public int getCurrentFile()
```

Returns:

the CurrentFile property

getFileExtension()

```
public java.lang.String getFileExtension()
```

Returns:

the FileExtension property

getFileNameBase()

```
public java.lang.String getFileNameBase()
```

Returns:

the FileNameBase property

getHeader()

```
public java.lang.String getHeader()
```

Get the header string that will be written at the beginning of each log file.

Returns:

the Header Property

getMaxFiles()

```
public int getMaxFiles()
```

Returns:

the MaxFiles property

getMaxFileSize()

```
public int getMaxFileSize()
```

Returns:

the MaxFileSize property

setHeader(String)

```
public void setHeader(java.lang.String header)
```

Set the constant header string that will be written at the beginning of every file, trace writing continues from the next line after the header is written. If `setHeader` is called after a file output has started, it will take effect from the next file to be written.

Usage:

```
tm = TraceManagerFactory.registerModule(this);
tw = newLogFileTraceWriter("trace", "log", 10, 1024*1024);
tw.setHeader(header);
tm.getTraceWriterManager().addTraceWriter(tw);
```

OutputStreamTraceWriter

OutputStreamTraceWriter wraps an output stream in a TraceWriter. This simplifies adding custom tracing classes that can co-exist with other TraceWriters.

Declaration

```
public final class OutputStreamTraceWriter extends BaseTraceWriter, on page 689

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.OutputStreamTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 722

Member Summary

Member summary	
Constructors	
	OutputStreamTraceWriter(int, OutputStream), on page 703 Default constructor which is auto-flushing
	OutputStreamTraceWriter(int, OutputStream, boolean), on page 703 Create an OutputStreamTraceWriter
Methods	
`protected void`	doClose(), on page 703
`protected void`	doFlush(), on page 703
`protected void`	doPrintln(String, int), on page 703
`java.io.OutputStream`	getOutputStream(), on page 704

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 689
close(), on page 691, doClose(), on page 692, flush(), on page 692, getDescription(), on page 692, getEnabled(), on page 692, getName(), on page 692, getTraceLevels(), on page 692, println(String, int), on page 693, setTraceLevels(int[]), on page 693, toString(), on page 693
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

OutputStreamTraceWriter(int, OutputStream)

```
public OutputStreamTraceWriter(int maxTraceLevel,
   java.io.OutputStreamoutputStream)
```

Default constructor which is auto-flushing

See Also:

Trace, on page 708

OutputStreamTraceWriter(int, OutputStream, boolean)

```
public OutputStreamTraceWriter(int maxTraceLevel,
   java.io.OutputStreamoutputStream,
   booleanautoFlush)
```

Create an OutputStreamTraceWriter

See Also:

Trace, on page 708

Methods

doClose()

```
protected void doClose()
```

Overrides:

`doClose` in class BaseTraceWriter, on page 689

doFlush()

```
protected void doFlush()
```

Overrides:

`doFlush` in class BaseTraceWriter, on page 689

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,  intmessageNumber)
```

Description copied from class: `com.cisco.services.tracing.BaseTraceWriter`

Must be implemented by the various TraceWriters extending BaseTraceWriter to provide the specific tracing functionality

Overrides:

`doPrintln` in class BaseTraceWriter, on page 689

getOutputStream()

```
public java.io.OutputStream getOutputStream()
```

Returns:

the output stream associated with the TraceWriter

SyslogTraceWriter

SyslogTraceWriter refines the BaseTraceWriter to allow tracing to syslog. Cisco syslog specification calls for sending low level traces to a syslog collector in the form of UDP messages. No buffering is done in this TraceWriter. The SyslogTraceWriter makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet.

Declaration

```
public final class SyslogTraceWriter extends BaseTraceWriter, on page 689

java.lang.Object
|
+--com.cisco.services.tracing.BaseTraceWriter
|
+--com.cisco.services.tracing.SyslogTraceWriter
```

All Implemented Interfaces

TraceWriter, on page 722

Member Summary

Member summary	
Constructors	
	SyslogTraceWriter(int, String), on page 705 Default SyslogTraceWriter with a max trace level of INFORMATIONAL
	SyslogTraceWriter(int, String, int), on page 705 SyslogTraceWriter with max trace level specified
	SyslogTraceWriter(int, String, int[]), on page 705 SyslogTraceWriter which takes an array of trace levels.
Methods	
`void`	doClose(), on page 706 Closes the socket

Member summary	
`protected void`	doPrintln(String, int), on page 706 The SyslogTraceWriter makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet.
`static void`	main(String[]), on page 706

Inherited member summary
Methods inherited from class BaseTraceWriter, on page 689
close(), on page 691, doClose(), on page 692, flush(), on page 692, getDescription(), on page 692, getEnabled(), on page 692, getName(), on page 692, getTraceLevels(), on page 692, println(String, int), on page 693, setTraceLevels(int[]), on page 693, toString(), on page 693
Methods inherited from class Object
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

SyslogTraceWriter(int, String)

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector)
```

Default SyslogTraceWriter with a max trace level of INFORMATIONAL

See Also:

Trace, on page 708

SyslogTraceWriter(int, String, int)

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector,
   intmaxTraceLevel)
```

SyslogTraceWriter with max trace level specified

See Also:

Trace, on page 708

SyslogTraceWriter(int, String, int[])

```
public SyslogTraceWriter(int port,
   java.lang.Stringcollector,
   int[]traceLevels)
```

SyslogTraceWriter which takes an array of trace levels.

See Also:

Trace, on page 708

Methods

doClose()

```
public void doClose()
```

Closes the socket

Overrides:

`doClose` in class BaseTraceWriter, on page 689

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
   intmessageNumber)
```

The `SyslogTraceWriter` makes an exception to the println() method in that it places a '\0' instead of a System specified line separator to terminate the message packet. The portion of the message after a '\r' or '\n' is ignored

Overrides:

`doPrintln` in class BaseTraceWriter, on page 689

main(String[])

```
public static void main(java.lang.String[] args)
```

TraceManagerFactory

The TraceManagerFactory class is a class by which applications obtain a TraceManager object. The TraceModule passed in the constructor is registered in a list. The list can be enumerated using the getModules() method.

Declaration

```
public class TraceManagerFactory

java.lang.Object
|
+--com.cisco.services.tracing.TraceManagerFactory
```

Member Summary

Member summary
Methods

Member summary	
`static java.util.Enumeration`	getModules(), on page 707 Returns an enumeration of the TraceModules registered with this factory.
`static TraceManager`	registerModule(TraceModule), on page 707 Returns an instance of a TraceManager object.
`static TraceManager`	registerModule(TraceModule, String[], TraceWriterManager), on page 707 Returns an instance of a TraceManager object.
`static TraceManager`	registerModule(TraceModule, TraceWriterManager), on page 708 Returns an instance of a TraceManager object.

Inherited member summary
Methods inherited from class `Object`
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods

getModules()

```
public static java.util.Enumeration getModules()
```

Returns an enumeration of the TraceModules registered with this factory.

registerModule(TraceModule)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module)
```

Returns an instance of a TraceManager object. The contained TraceWriterManager will not have any default TraceWriters.

registerModule(TraceModule, String[], TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module,
   java.lang.String[]subFacilities,
   com.cisco.services.tracing.TraceWriterManagertraceWriterManager)
```

Returns an instance of a TraceManager object. Trace output will be redirected to the TraceWriterManager object specified.

registerModule(TraceModule, TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
   registerModule(com.cisco.services.tracing.TraceModule module,
   com.cisco.services.tracing.TraceWriterManagertraceWriterManager)
```

Returns an instance of a TraceManager object. Trace output will be redirected to the TraceWriterManager object specified.

Services Tracing Interface Hierarchy

The following interface hierarchy is contained in the com.cisco.services.tracing package.

com.cisco.services.tracing.Trace, on page 708
   com.cisco.services.tracing.ConditionalTrace, on page 715
com.cisco.services.tracing.UnconditionalTrace, on page 716

com.cisco.services.tracing.TraceManager, on page 717

com.cisco.services.tracing.TraceModule, on page 721

com.cisco.services.tracing.TraceWriter, on page 722

com.cisco.services.tracing.TraceWriterManager, on page 725

Trace

The Trace interface defines the methods that allow application tracing. Trace also defines the standard trace types as specified by Syslog Trace Logging.Syslog currently defines 8 levels of trace. The severity of the message is indicated in the trace as a number ranging between [0-7] (0 and 7 included). Currently 7 is HIGHEST_LEVEL and 0 is the LOWEST_LEVEL trace. All 8 levels are predefined here as static int types for reference in tracing sub-system implementations.

The severities traced are as follows:

0 = EMERGENCIES System unusable

1 = ALERTS Immediate action needed

2 = CRITICAL Critical conditions

3 = ERROR Error conditions

4 = WARNING Warning conditions

5 = NOTIFICATION Normal but significant condition

6 = INFORMATIONAL Informational messages only

7 = DEBUGGING Debugging messages

Declaration

```
public interface Trace
```

All Known Subinterfaces

ConditionalTrace, on page 715UnconditionalTrace, on page 716

Member Summary

Member summary	
Fields	
`static int`	ALERTS, on page 711 The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed) Syslog severity level = 1
`static java.lang.String`	ALERTS_TRACE_NAME, on page 711 String descriptor for ALERTS trace level
`static int`	CRITICAL, on page 685 A critical failure, the application cannot accomplish the tasks required due to this failure, e.g.: the application cant open the database to read the device list Syslog severity level = 2
`static java.lang.String`	CRITICAL_TRACE_NAME, on page 712 String descriptor for CRITICAL trace level
`static int`	DEBUGGING, on page 712 Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled Syslog severity level = 7
`static java.lang.DEBUGGING_TRACE_NAME, on page 712String`	DEBUGGING_TRACE_NAME, on page 712 String descriptor for the DEBUGGING trace level
`static int`	EMERGENCIES, on page 712 Emergency situation, a system shutdown is necessary Syslog severity level = 0
`static java.lang.String`	EMERGENCIES_TRACE_NAME, on page 712 String descriptor for EMERGENCIES trace level

Member summary	
`static int`	ERROR, on page 712 An error condition of some kind has occurred and the user needs to understand the nature of that failure Syslog severity level = 3
`static java.lang.String`	ERROR_TRACE_NAME, on page 712 String descriptor for ERROR trace level
`static int`	HIGHEST_LEVEL, on page 712 The highest trace level, currently this is DEBUGGING with a trace level of 7
`static int`	INFORMATIONAL, on page 713 Information of some form not relating to errors, warnings, audit, or debug Syslog severity level = 6
`static java.lang.String`	INFORMATIONAL_TRACE_NAME, on page 713 String descriptor for INFORMATIONAL trace level
`static int`	LOWEST_LEVEL, on page 713 The lowest trace level, currently this is EMERGENCIES with a trace level of 0
`static int`	NOTIFICATION, on page 713 Notification denotes a normal but significant condition Syslog severity level = 5
`static java.lang.String`	NOTIFICATION_TRACE_NAME, on page 713 String descriptor for NOTIFICATION trace level
`static int`	WARNING, on page 713 Warning that a problem of some form exists but is not keeping the application from completing its tasks Syslog severity level = 4
`static java.lang.String`	WARNING_TRACE_NAME, on page 713 String descriptor for WARNING trace level
Methods	
`java.lang.String`	getName(), on page 713 Returns the name of this Trace object.

Member summary	
`java.lang.String`	getSubFacility(), on page 714 Returns the subFacility of trace
`int`	getType(), on page 714 Returns the type of trace.
`boolean`	isEnabled(), on page 714 Returns the state of this Trace object.
`void`	println(Object), on page 714 Prints the string returned by the Object.toString() method and terminates the line as defined by the system.
`void`	println(String), on page 714 Prints a message in the same format as Trace.print() and terminates the line as defined by the system.
`void`	println(String, Object), on page 714 Prints the string returned by the Object.toString() method and terminates the line as defined by the system.
`void`	println(String, String), on page 715 Prints a message in the same format as Trace.print() and terminates the line as defined by the system.
`void`	setDefaultMnemonic(String), on page 715 Sets a default mnemonic for all messages printed out to this trace.

Fields

ALERTS

```
public static final int ALERTS
```

The application will continue working on the tasks but all functions may not be operational (one or more devices in the list are not accessible but others in the list can be accessed)

Syslog severity level = 1

ALERTS_TRACE_NAME

```
public static final java.lang.String ALERTS_TRACE_NAME
```

String descriptor for ALERTS trace level

CRITICAL

```
public static final int CRITICAL
```

A critical failure, the application cannot accomplish the tasks required due to this failure, e.g.: the application cant open the database to read the device list

Syslog severity level = 2

CRITICAL_TRACE_NAME

```
public static final java.lang.String CRITICAL_TRACE_NAME
```

String descriptor for CRITICAL trace level

DEBUGGING

```
public static final int DEBUGGING
```

Very detailed information regarding errors or processing status that is only generated when DEBUG mode has been enabled

Syslog severity level = 7

DEBUGGING_TRACE_NAME

```
public static final java.lang.String DEBUGGING_TRACE_NAME
```

String descriptor for the DEBUGGING trace level

EMERGENCIES

```
public static final int EMERGENCIES
```

Emergency situation, a system shutdown is necessary

Syslog severity level = 0

EMERGENCIES_TRACE_NAME

```
public static final java.lang.String EMERGENCIES_TRACE_NAME
```

String descriptor for EMERGENCIES trace level

ERROR

```
public static final int ERROR
```

An error condition of some kind has occurred and the user needs to understand the nature of that failure

Syslog severity level = 3

ERROR_TRACE_NAME

```
public static final java.lang.String ERROR_TRACE_NAME
```

String descriptor for ERROR trace level

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

The highest trace level, currently this is DEBUGGING with a trace level of 7

INFORMATIONAL

```
public static final int INFORMATIONAL
```

Information of some form not relating to errors, warnings, audit, or debug

Syslog severity level = 6

INFORMATIONAL_TRACE_NAME

```
public static final java.lang.String INFORMATIONAL_TRACE_NAME
```

String descriptor for INFORMATIONAL trace level

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

The lowest trace level, currently this is EMERGENCIES with a trace level of 0

NOTIFICATION

```
public static final int NOTIFICATION
```

Notification denotes a normal but significant condition

Syslog severity level = 5

NOTIFICATION_TRACE_NAME

```
public static final java.lang.String NOTIFICATION_TRACE_NAME
```

String descriptor for NOTIFICATION trace level

WARNING

```
public static final int WARNING
```

Warning that a problem of some form exists but is not keeping the application from completing its tasks

Syslog severity level = 4

WARNING_TRACE_NAME

```
public static final java.lang.String WARNING_TRACE_NAME
```

String descriptor for WARNING trace level

Methods

getName()

```
public java.lang.String getName()
```

Returns:

the name of this Trace object

getSubFacility()

```
public java.lang.String getSubFacility()
```

Returns:

the trace subFacility type

getType()

```
public int getType()
```

Returns:

the type of trace as specified in Syslog. DEBUGGING, INFORMATIONAL, WARNING, etc.

isEnabled()

```
public boolean isEnabled()
```

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Returns:

true if tracing is enabled, false otherwise

See Also

ConditionalTrace, on page 715

println(Object)

```
public void println(java.lang.Object object)
```

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Parameters:

`object` - the object to be printed

println(String)

```
public void println(java.lang.String message)
```

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Parameters:

`message` - the message to be printed

println(String, Object)

```
public void println(java.lang.String mnemonic,
   java.lang.Objectobject)
```

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Parameters:

`object` - the object to be printed

`mnemonic` - the mnemonic mapped to message to be printed

println(String, String)

```
public void println(java.lang.String mnemonic,
   java.lang.Stringmessage)
```

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Parameters:

`message` - the message to be printed

`mnemonic` - the mnemonic mapped to message to be printed

setDefaultMnemonic(String)

```
public void setDefaultMnemonic(java.lang.String mnemonic)
```

Sets a default mnemonic for all messages printed out to this trace.

Parameters:

`mnemonic,` - a mnemonic string

ConditionalTrace

The ConditionalTrace interface extends the Trace interface and defines the methods that allow enabling and disabling of tracing for this particular condition.

Typically, applications obtain one ConditionalTrace object for each condition that they need to trace under certain circumstances but not always (for example, AUDIT, INFO, and so on).

Declaration

```
public interface ConditionalTrace extends Trace, on page 708
```

All Superinterfaces

Trace, on page 708

Member Summary

Member summary	
Methods	
`void`	disable(), on page 716 Disables this condition for tracing.
`void`	enable(), on page 716 Enables this condition for tracing.

Inherited member summary
Fields inherited from interface Trace, on page 708
ALERTS, on page 711, ALERTS_TRACE_NAME, on page 711, CRITICAL, on page 685, CRITICAL_TRACE_NAME, on page 712, DEBUGGING, on page 712, DEBUGGING_TRACE_NAME, on page 712, EMERGENCIES, on page 712, EMERGENCIES_TRACE_NAME, on page 712, ERROR, on page 712, ERROR_TRACE_NAME, on page 712, HIGHEST_LEVEL, on page 712, INFORMATIONAL, on page 713, INFORMATIONAL_TRACE_NAME, on page 713, LOWEST_LEVEL, on page 713, NOTIFICATION, on page 713, NOTIFICATION_TRACE_NAME, on page 713, WARNING, on page 713, WARNING_TRACE_NAME, on page 713
Methods inherited from interface Trace, on page 708
getName(), on page 713, getSubFacility(), on page 714, getType(), on page 714, isEnabled(), on page 714, println(Object), on page 714, println(String), on page 714, println(String, Object), on page 714, println(String, String), on page 715, setDefaultMnemonic(String), on page 715

Methods

disable()

```
public void disable()
```

Disables this condition for tracing.

enable()

```
public void enable()
```

Enables this condition for tracing.

UnconditionalTrace

The UnconditionalTrace interface extends the Trace interface. Note that because this object extends Trace, its state is enabled by default and it may not be changed.

Typically, applications would obtain one UnconditionalTrace object per each condition that they need to trace always under any circumstances (such as, ERROR, FATAL, and so on).

Declaration

```
public interface UnconditionalTrace extends Trace, on page 708
```

All Superinterfaces

Trace, on page 708

Member Summary

Inherited Member summary
Fields inherited from interface Trace, on page 708
ALERTS, on page 711, ALERTS_TRACE_NAME, on page 711, CRITICAL, on page 685, CRITICAL_TRACE_NAME, on page 712, DEBUGGING, on page 712, DEBUGGING_TRACE_NAME, on page 712, EMERGENCIES, on page 712, EMERGENCIES_TRACE_NAME, on page 712, ERROR, on page 712, ERROR_TRACE_NAME, on page 712, HIGHEST_LEVEL, on page 712, INFORMATIONAL, on page 713, INFORMATIONAL_TRACE_NAME, on page 713, LOWEST_LEVEL, on page 713, NOTIFICATION, on page 713, NOTIFICATION_TRACE_NAME, on page 713, WARNING, on page 713, WARNING_TRACE_NAME, on page 713
Methods inherited from interface Trace, on page 708
getName(), on page 713, getSubFacility(), on page 714, getType(), on page 714, isEnabled(), on page 714, println(Object), on page 714, println(String), on page 714, println(String, Object), on page 714, println(String, String), on page 715, setDefaultMnemonic(String), on page 715

TraceManager

The TraceManager interface defines the methods that allow applications trace management.

Typically, an application obtains only one TraceManager object. All Trace objects are created by default: Predefined Trace in accordance with Syslog definitions are:

```
ConditionalTraces:INFORMATIONAL, DEBUGGING, NOTIFICATION, WARNING
UnconditionalTraces:ERROR, CRITICAL, ALERTS, EMERGENCIES
```

Facilities/Sub-Facilities:

- **Facility**—A code consisting of two or more uppercase letters that indicate the facility to which the message refers. A facility can be a hardware device, a protocol, or a module of the system software.
- **SubFacility**—A code consisting of two or more uppercase letters that indicate the sub-facility to which the message refers. A sub-facility can be a hardware device component, a protocol unit, or a sub-module of the system software.

By default all 8 Conditional and UnConditional Traces are created for the Facility and 8 for each of the subFacilities In order to use the DEBUGGING trace for the parent FACILITY, for example, the application needs to use the getConditionalTrace(“DEBUGGING”) method of this object.

In order to use the DEBUGGING trace for the SUBFACILITY, for example, the application needs to use the getConditionalTrace(SUBFACILITY + “_” + “DEBUGGING”) method of this object or use the getConditionalTrace(SUBFACILITY, “DEBUGGING”) method.

System wide TraceWriterManager is set through the setTraceWriterManager method provided by this interface.

The Trace Manager object also allows the application to enable or disable tracing for all trace through the enableAll() and disableAll() methods.

Declaration

```
public interface TraceManager
```

Member Summary

Member summary	
Methods	
`void`	addSubFacilities(String[]), on page 719 Sets a set of subFacilities for this TraceManager/Facility.
`void`	addSubFacility(String), on page 719 Adds a single subFacility for this TraceManager/Facility.
`void`	disableAll(), on page 719 Disables tracing for all Trace objects managed by this TraceManager.
`void`	disableTimeStamp(), on page 720 Disables prefixing a time stamp for every message printed by this TraceManager.
`void`	enableAll(), on page 720 Enables tracing for all Trace objects managed by this TraceManager.
`void`	enableTimeStamp(), on page 720 Enables prefixing a time stamp for every message printed by this TraceManager.
`ConditionalTrace`	getConditionalTrace(int), on page 720 Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.
`ConditionalTrace`	getConditionalTrace(String, int), on page 720 Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility
`java.lang.String`	getName(), on page 720 Returns the Facility name for this TraceManager.
`java.lang.String[]`	getSubFacilities(), on page 720 Returns the subFacility names for this TraceManager/Facility.

Member summary	
`java.util.Enumeration`	getTraces(), on page 720 Returns an enumeration of the Trace objects managed by this TraceManager.
`TraceWriterManager`	getTraceWriterManager(), on page 720 Returns the TraceWriter used by this TraceManager.
`UnconditionalTrace`	getUnconditionalTrace(int), on page 720 Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.
`UnconditionalTrace`	getUnconditionalTrace(String, int), on page 721 Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility
`void`	removeTrace(Trace), on page 721 Removes a Trace object given an object.
`void`	setSubFacilities(String[]), on page 721 Sets a set of subFacilities for this TraceManager/Facility.
`void`	setSubFacility(String), on page 721 Adds a single subFacility for this TraceManager/Facility.
`void`	setTraceWriterManager(TraceWriterManager), on page 721 Sets the TraceWriter to be used by this TraceManager.

Methods

addSubFacilities(String[])

```
public void addSubFacilities(java.lang.String[] names)
```

Sets a set of subFacilities for this TraceManager/Facility.

addSubFacility(String)

```
public void addSubFacility(java.lang.String name)
```

Adds a single subFacility for this TraceManager/Facility.

disableAll()

```
public void disableAll()
```

Disables tracing for all Trace objects managed by this TraceManager.

disableTimeStamp()

```
public void disableTimeStamp()
```

Disables prefixing a time stamp for every message printed by this TraceManager.

enableAll()

```
public void enableAll()
```

Enables tracing for all Trace objects managed by this TraceManager.

enableTimeStamp()

```
public void enableTimeStamp()
```

Enables prefixing a time stamp for every message printed by this TraceManager.

getConditionalTrace(int)

```
public com.cisco.services.tracing.ConditionalTrace
   getConditionalTrace(int severity)
```

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.

getConditionalTrace(String, int)

```
public com.cisco.services.tracing.ConditionalTrace
   getConditionalTrace(java.lang.String subFacility,
   intseverity)
```

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility

getName()

```
public java.lang.String getName()
```

Returns the Facility name for this TraceManager.

getSubFacilities()

```
public java.lang.String[] getSubFacilities()
```

Returns the subFacility names for this TraceManager/Facility.

getTraces()

```
public java.util.Enumeration getTraces()
```

Returns an enumeration of the Trace objects managed by this TraceManager.

getTraceWriterManager()

```
public com.cisco.services.tracing.TraceWriterManager getTraceWriterManager()
```

Returns the TraceWriter used by this TraceManager.

getUnconditionalTrace(int)

```
public com.cisco.services.tracing.UnconditionalTrace getUnconditionalTrace(int severity)
```

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.

getUnconditionalTrace(String, int)

```
public com.cisco.services.tracing.UnconditionalTrace
   getUnconditionalTrace(java.lang.String subFacility,
   intseverity)
```

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility

removeTrace(Trace)

```
public void removeTrace(com.cisco.services.tracing.Trace tc)
```

Removes a Trace object given an object.

setSubFacilities(String[])

```
public void setSubFacilities(java.lang.String[] names)
```

Deprecated and replaced with `TraceManager.addSubFacilities` method

Sets a set of subFacilities for this TraceManager/Facility.

setSubFacility(String)

```
public void setSubFacility(java.lang.String name)
```

Deprecated and replaced with `TraceManager.addSubFacility` method

Adds a single subFacility for this TraceManager/Facility.

setTraceWriterManager(TraceWriterManager)

```
public void setTraceWriterManager(com.cisco.services.tracing.TraceWriterManager twm)
```

Sets the TraceWriter to be used by this TraceManager.

TraceModule

The TraceModule interface serves two purposes. First, it allows applications to discover the TraceManager object used by other packages that they use. Second, applications that register with the TraceManagerFactory must identify themselves by implementing this interface.

Declaration

```
public interface TraceModule
```

All Known Subinterfaces

com.cisco.jtapi.extensions.CiscoJtapiPeer

Member Summary

Member summary	
Methods	
`TraceManager`	getTraceManager(), on page 722 Returns the TraceManager that an object is using for tracing.
`java.lang.String`	getTraceModuleName(), on page 722 Returns the module name.

Methods

getTraceManager()

```
public com.cisco.services.tracing.TraceManager getTraceManager()
```

Returns the TraceManager that an object is using for tracing.

getTraceModuleName()

```
public java.lang.String getTraceModuleName()
```

Returns the module name.

TraceWriter

The TraceWriter interface abstracts the details of trace message output. The TraceWriter uses its enabled method to advertise whether or not the print and println methods will have any effect. Users of TraceWriter should use the value returned by the getEnabled method as an indication of whether they should invoke the print and println methods at all.

Declaration

```
public interface TraceWriter
```

All Known Subinterfaces

TraceWriterManager, on page 725

All Known Implementing Classes

BaseTraceWriter, on page 689

Member Summary

Member summary	
Methods	
`void`	close(), on page 723 Releases any resources associated by this TraceWriter .
`void`	flush(), on page 723 Forces output of any messages that have been printed using the println method
`java.lang.String`	getDescription(), on page 723
`boolean`	getEnabled(), on page 724 Returns whether the println method will print anything or not.
`java.lang.String`	getName(), on page 724
`int[]`	getTraceLevels(), on page 724
`void`	println(String, int), on page 724 Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream.
`void`	setTraceLevels(int[]), on page 724 set the trace levels that will be traced by this TraceWriter

Methods

close()

```
public void close()
```

Releases any resources associated by this TraceWriter.

flush()

```
public void flush()
```

Forces output of any messages that have been printed using the println method

getDescription()

```
public java.lang.String getDescription()
```

Returns:

a short description of this TraceWriter

getEnabled()

```
public boolean getEnabled()
```

Returns whether the println method will print anything or not. A closed `TraceWriter` will always return `false` from this method.

Returns:

true if this TraceWriter is enabled, false if not

getName()

```
public java.lang.String getName()
```

Returns:

the name of this TraceWriter

getTraceLevels()

```
public int[] getTraceLevels()
```

Returns:

the array of trace levels that will be traced by this TraceWriter

println(String, int)

```
public void println(java.lang.String message,
   intseverity)
```

Prints the specified string followed by a carriage return The concrete TraceWriter class will use the severity to block out messages from a particular stream. Each trace writer has a notion of the highest level trace it traces

Parameters:

`message` - the string to print

`severity` - of the trace.

See Also

Trace, on page 708

setTraceLevels(int[])

```
public void setTraceLevels(int[] levels)
```

set the trace levels that will be traced by this TraceWriter

Parameters:

`int[]` - levels

See Also

Trace, on page 708

TraceWriterManager

TraceWriterManager contains the list of TraceWriter objects that are used to implement the tracing. The list is populated at startup from the switches in a .ini file. A LogFileTraceWriter, a ConsoleTraceWriter, and a SyslogTraceWriter are available. Users can override the existing TraceWriters by setting a user implemented TraceWriter[] or adding to the existing TraceWriters. This makes it possible to add other TraceWriters that can function along with existing trace writers.

Declaration

```
public interface TraceWriterManager extends TraceWriter, on page 722
```

All Superinterfaces

TraceWriter, on page 722

Member Summary

Member summary	
Methods	
`void`	addTraceWriter(TraceWriter), on page 726 Add another TraceWriter to the array
`TraceWriter[]`	getTraceWriters(), on page 726
`void`	removeTraceWriter(TraceWriter), on page 726 Remove the TraceWriter from the array in the manager
`void`	setTraceWriters(TraceWriter[]), on page 726 Implementations can use this method to override or enhance the provided TraceWriters

Inherited member summary
Methods inherited from interface TraceWriter, on page 722
close(), on page 723, flush(), on page 723, getDescription(), on page 723, getEnabled(), on page 724, getName(), on page 724, getTraceLevels(), on page 724, println(String, int), on page 724, setTraceLevels(int[]), on page 724

Methods

addTraceWriter(TraceWriter)

```
public voidaddTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Add another TraceWriter to the array

Parameters:

`TraceWriter` - to be added to the list

getTraceWriters()

```
public com.cisco.services.tracing.TraceWriter[] getTraceWriters()
```

Returns:

the array of TraceWriters in the manager

removeTraceWriter(TraceWriter)

```
public voidremoveTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Remove the TraceWriter from the array in the manager

setTraceWriters(TraceWriter[])

```
public voidsetTraceWriters(com.cisco.services.tracing.TraceWriter[] traceWriters)
```

Implementations can use this method to override or enhance the provided TraceWriters

Parameters:

`set` - the array of TraceWriters.

Tracing Implementation Class Hierarchy

The following tracing implementation class hierarchy is contained in the com.cisco.services.tracing.implementation package.

```
java.lang.Object
   com.cisco.services.tracing.implementation.TraceImpl, on page 727 (implements
                                                 com.cisco.services.tracing.Trace)
   com.cisco.services.tracing.implementation.ConditionalTraceImpl, on page 729 (implements
                           com.cisco.services.tracing.ConditionalTrace)
   com.cisco.services.tracing.implementation.UnconditionalTraceImpl, on page 730 (implements

                         com.cisco.services.tracing.UnconditionalTrace)
com.cisco.services.tracing.implementation.TraceManagerImpl, on page 731 (implements
                                    com.cisco.services.tracing.TraceManager)
com.cisco.services.tracing.implementation.TraceWriterManagerImpl, on page 735 (implements
                               com.cisco.services.tracing.TraceWriterManager)
```

TraceImpl

Declaration

public abstract class **TraceImpl**

extends java.lang.Object

implements Trace

All Implemented Interfaces

Trace, on page 708

Methods

println

public final void **println**(java.lang.String message)

Description copied from interface: Trace

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

message - the message to be printed

println

public final void println(java.lang.String mnemonic, java.lang.String message)

Description copied from interface: Trace

Prints a message in the same format as Trace.print() and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

mnemonic - the mnemonic mapped to message to be printed

message - the message to be printed

println

public final void **println**(java.lang.Object object)

Description copied from interface: Trace

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

object - the object to be printed

println

public final void **println**(java.lang.String mnemonic, java.lang.Object object)

Description copied from interface: Trace

Prints the string returned by the Object.toString() method and terminates the line as defined by the system.

Specified by:

println in interface Trace

Parameters:

mnemonic - the mnemonic mapped to message to be printed

object - the object to be printed

getName

public final java.lang.String **getName**()

Description copied from interface: Trace

Returns the name of this Trace object.

Specified by:

getName in interface Trace

Returns:

the name of this Trace object

setDefaultMnemonic

public final void **setDefaultMnemonic**(java.lang.String mnemonic)

Description copied from interface: Trace

Sets a default mnemonic for all messages printed out to this trace.

Specified by:

setDefaultMnemonic in interface Trace

Parameters:

mnemonic - a mnemonic string

getType

public int **getType**()

Description copied from interface: Trace

Returns the type of trace.

Specified by:

getType in interface Trace

Returns:

the trace severity as specified in Syslog. DEBUGGING, INFORMATIONAL, WARNING, etc.

getSubFacility

public java.lang.String **getSubFacility**()

Description copied from interface: Trace

Returns the subFacility of trace

Specified by:

getSubFacility in interface Trace

Returns:

the trace subFacility type

Inherited Methods

isEnabled

ConditionalTraceImpl

Declaration

public final class ConditionalTraceImpl

extends TraceImpl

implements ConditionalTrace

All Implemented Interfaces

ConditionalTrace, Trace

Methods

enable

public void **enable**()

Description copied from interface: ConditionalTrace

Enables this condition for tracing.

Specified by:

enable in interface ConditionalTrace

disable

public void **disable**()

Description copied from interface: ConditionalTrace

Disables this condition for tracing.

Specified by:

disable in interface ConditionalTrace

isEnabled

public boolean **isEnabled**()

Description copied from interface: Trace

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Specified by:

isEnabled in interface Trace

Returns:

true if tracing is enabled, false otherwise

See Also:

ConditionalTrace

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

UnconditionalTraceImpl

Declaration

public final class UnconditionalTraceImpl

extends TraceImpl

implements UnconditionalTrace

All Implemented Interfaces

Trace, UnconditionalTrace

Methods

isEnabled

public boolean **isEnabled**()

Description copied from interface: Trace

Returns the state of this Trace object. By default, Trace objects are enabled, that is, println() method will always trace. The state may not be changed through this interface, however, this object may implement additional interfaces that allow the state to be changed.

Specified by:

isEnabled in interface Trace

Returns:

true if tracing is enabled, false otherwise

See Also:

ConditionalTrace

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

TraceManagerImpl

The TraceManagerImpl class implements the TraceManager interface.

Declaration

public class **TraceManagerImpl** extends java.lang.Object

java.lang.Object

|

+--com.cisco.services.tracing.implementation.TraceManagerImpl

All Implemented Interfaces

TraceManager, on page 717

Constructors

public **TraceManagerImpl**(java.lang.StringmoduleName, java.lang.String[]subFacilities, TraceWriterManagertraceWriterManager)

public **TraceManagerImpl**(java.lang.StringmoduleName, TraceWriterManagertraceWriterManager)

Methods

getConditionalTrace

```
public ConditionalTrace getConditionalTrace(intseverity)
```

Description copied from interface: TraceManager

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition.

Specified by:

`getConditionalTrace` in interface `TraceManager`

getConditionalTrace

```
public ConditionalTrace getConditionalTrace(java.lang.StringsubFacility,
   intseverity)
```

Description copied from interface: TraceManager

Creates a new ConditionalTrace object or obtains an existing ConditionalTrace object for this condition and subFacility

Specified by:

`getConditionalTrace` in interface `TraceManager`

getUnconditionalTrace

```
public UnconditionalTrace getUnconditionalTrace(intseverity)
```

Description copied from interface: TraceManager

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition.

Specified by:

`getUnconditionalTrace` in interface `TraceManager`

getUnconditionalTrace

```
public UnconditionalTrace getUnconditionalTrace(java.lang.StringsubFacility,
   intseverity)
```

Description copied from interface: TraceManager

Creates a new UnconditionalTrace object or obtains an existing UnconditionalTrace object for this condition and subFacility

Specified by:

`getUnconditionalTrace` in interface `TraceManager`

getTraceWriterManager

```
public TraceWriterManager getTraceWriterManager()
```

Description copied from interface: TraceManager

Returns the TraceWriter used by this TraceManager.

Specified by:

`getTraceWriterManager` in interface `TraceManager`

setTraceWriterManager

```
public void setTraceWriterManager(TraceWriterManagerout)
```

Description copied from interface: TraceManager

Sets the TraceWriter to be used by this TraceManager.

Specified by:

`setTraceWriterManager` in interface `TraceManager`

removeTrace

```
public void removeTrace(Tracetc)
```

Description copied from interface: TraceManager

Removes a Trace object given an object.

Specified by:

`removeTrace` in interface `TraceManager`

getTraces

```
public java.util.Enumeration getTraces()
```

Description copied from interface: TraceManager

Returns an enumeration of the Trace objects managed by this TraceManager.

Specified by:

`getTraces` in interface `TraceManager`

enableAll

```
public void enableAll()
```

Description copied from interface: TraceManager

Enables tracing for all Trace objects managed by this TraceManager.

Specified by:

`enableAll` in interface `TraceManager`

disableAll

```
public void disableAll()
```

Description copied from interface: TraceManager

Disables tracing for all Trace objects managed by this TraceManager.

Specified by:

`disableAll` in interface `TraceManager`

getName

```
public java.lang.String getName()
```

Description copied from interface: TraceManager

Returns the Facility name for this TraceManager.

Specified by:

`getName` in interface `TraceManager`

enableTimeStamp

```
public void enableTimeStamp()
```

Description copied from interface: TraceManager

Enables prefixing a time stamp for every message printed by this TraceManager.

Specified by:

`enableTimeStamp` in interface `TraceManager`

disableTimeStamp

```
public void disableTimeStamp()
```

Description copied from interface: TraceManager

Disables prefixing a time stamp for every message printed by this TraceManager.

Specified by:

`disableTimeStamp` in interface `TraceManager`

getSubFacilities

```
public java.lang.String[] getSubFacilities()
```

Returns the subFacility names for this TraceManager/Facility.

Specified by:

`getSubFacilities` in interface `TraceManager`

addSubFacilities

```
public void addSubFacilities(java.lang.String[]names)
```

Adds subFacilities for this TraceManager/Facility.

Specified by:

`addSubFacilities` in interface `TraceManager`

addSubFacility

```
public void addSubFacility(java.lang.Stringname)
```

Adds a subFacility for this TraceManager/Facility.

Specified by:

`addSubFacility` in interface `TraceManager`

Deprecated

getSubFacilities(java.lang.String[]names)

Replaced by addSubFacilties(String[]).

setSubFacility(java.lang.Stringname)

Replaced by addSubFacility(String).

Inherited Methods

Inherited methods from class java.lang.Object are: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

TraceWriterManagerImpl

TraceWriterManager contains the list of TraceWriter objects that are used to implement the tracing. The list is populated at startup from the switches in a .ini file. A LogFileTraceWriter, a ConsoleTraceWriter, and a SyslogTraceWriter are available. Users can override the existing TraceWriters by setting a user implemented TraceWriter[] or adding to the existing TraceWriters. This makes it possible to add other traceWriters that can function along with exisiting trace writers.

Note Methods inherited from class java.lang.Object are clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait.

Declaration

public class TraceWriterManagerImpl extends java.lang.Object implements TraceWriterManager

java.lang.Object

com.cisco.services.tracing.implementation.TraceWriterManagerImpl

All Implemented Interfaces

TraceWriter, TraceWriterManager

Constructors

TraceWriterManagerImpl

```
public TraceWriterManagerImpl()
```

Creates a TraceWriterManagerImpl with a zero length TraceWriter array .

Methods

setTraceWriters

```
public void setTraceWriters(TraceWriter[]traceWriters)
```

Overrides the existing TraceWriters with a new user supplied set .

Specified by:

`setTraceWriters` in interface `TraceWriterManager`

Parameters:

traceWriters - An array of TraceWriters.

getTraceWriters

```
public TraceWriter[] getTraceWriters()
```

Returns the array of TraceWriters currently in use .

Specified by:

`getTraceWriters` in interface `TraceWriterManager`

Returns:

The array of TraceWriters in the manager.

addTraceWriter

```
public void addTraceWriter(TraceWritertw)
```

Add this TraceWriter to the array of trace writers

Specified by:

`addTraceWriter` in interface `TraceWriterManager`

Parameters:

tw - TraceWriter to be added to the list

removeTraceWriter

```
public void removeTraceWriter(TraceWritertw)
```

Remove the Tracewriter from the array of trace writers.

Specified by:

`removeTraceWriter` in interface `TraceWriterManager`

println

```
public void println(java.lang.Stringmessage,  intseverity)
```

All traces invoke this method. A trace supplies its severity along with the message. Traces below the threshold severity of the TraceWriter are allowed. Eg. If the Threshhold severity is set to INFORMATIONAL (level = 6) DEBUG traces will not be passed by the TraceWriter. The severity level is set in the constructor of the TraceWriter

Specified by:

println in interface TraceWriter

Parameters:

`message` - The string to print

`severity` - The severity of the trace.

See Also:

Trace

Flush

public void flush()

Description copied from interface: TraceWriter

Forces output of any messages that have been printed using the println method

Specified by:

`flush` in interface `TraceWriter`

close

```
public void close()
```

Description copied from interface: TraceWriter

Releases any resources associated by this TraceWriter.

Specified by:

`close` in interface `TraceWriter`

getEnabled

```
public boolean getEnabled()
```

Returns true if any one of the underlying TraceWriter is enabled, else returns false.

Specified by:

`getEnabled` in interface `TraceWriter`

Returns:

`True` if this TraceWriter is enabled, `false` if not.

getName

```
public java.lang.String getName()
```

Specified by:

`getName` in interface `TraceWriter`

Returns:

The name of this TraceWriter.

getDescription

```
public java.lang.String getDescription()
```

Specified by:

`getDescription` in interface `TraceWriter`

Returns:

A short description of this TraceWriter.

setTraceLevels

```
public void setTraceLevels(int[]levels)
```

The TraceWriterManager does nothing for this method .

Specified by:

`setTraceLevels` in interface `TraceWriter`

Parameters:

Levels - Array of trace levels.

See Also:

Trace

getTraceLevels

```
public int[] getTraceLevels()
```

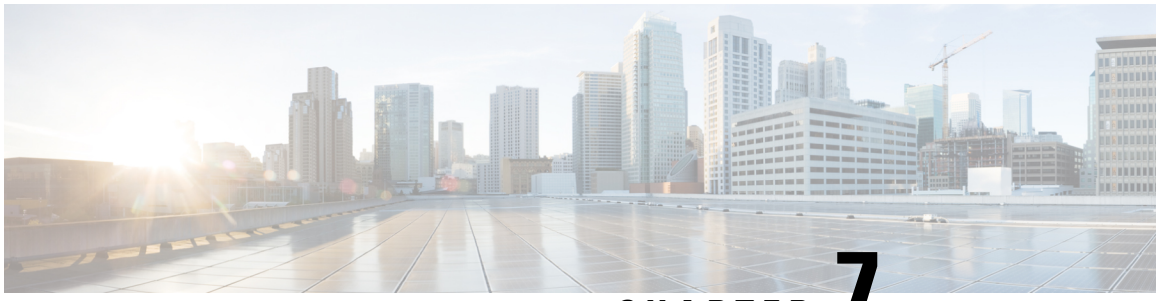
The TraceWriterManager returns a null, as the traceLevel is maintained at the individual TraceWriter .

Specified by:

`getTraceLevels` in interface `TraceWriter`

Returns:

`null`



CHAPTER 7

Cisco Unified JTAPI Examples

This chapter provides the source code for makecall, the Cisco Unified JTAPI program that is used to test the JTAPI installation. The makecall program comprises a series of programs that were written in Java by using the Cisco Unified JTAPI implementation.

For instructions on how to invoke makecall, see Running makecall, on page 753.

The Cisco Unified JTAPI Test tool can also be used to review message examples and test JTAPI features and functions. For details, refer http://developer.cisco.com/web/jtapi/docs.

MakeCall.java

```
/** * makecall.java
 *
 * Copyright Cisco Systems,  Inc.
 *
 * Performance-testing application (first pass) for Cisco JTAPI
 * implementation.
 *
 * Known problems:
 *
 * Due to synchronization problems between Actors,  calls may
 * not be cleared when this application shuts down.
 *
 */

//import com.ms.wfc.app.*;
import java.util.*;
import javax.telephony.*;
import javax.telephony.events.*;
import com.cisco.cti.util.Condition;
```

```
public class makecall extends TraceWindow implements ProviderObserver
{
  Vectoractors  =  new Vector ();
  ConditionconditionInService  =  new Condition ();
  Providerprovider;

  public makecall ( String [] args ) {

    super ( "makecall" + ": "+ new CiscoJtapiVersion());
    try
    {

      println ( "Initializing Jtapi" );
      int curArg  =  0;
      String providerName  =  args[curArg++];
      String login  =  args[curArg++];
      String passwd  =  args[curArg++];
      int actionDelayMillis  =  Integer.parseInt ( args[curArg++] );
      String src  =  null;
      String dest  =  null;

      JtapiPeer peer  =  JtapiPeerFactory.getJtapiPeer ( null );
      if ( curArg < args.length )
      {

        String providerString  =  providerName + ";login = " + login + ";passwd
= " + passwd;
        println ( "Opening " + providerString + "...\n" );
        provider  =  peer.getProvider ( providerString );
        provider.addObserver ( this );
        conditionInService.waitTrue ();

        println ( "Constructing actors" );

        for ( ; curArg < args.length; curArg++ )
        {
          if ( src  =  =  null )
          {
            src  =  args[curArg];
          }
          else
          {
            dest  =  args[curArg];
            Originator originator  =  new Originator ( provider.getAddress ( src
 ),
               dest,  this,  actionDelayMillis );
            actors.addElement ( originator );
            actors.addElement (
              new Receiver ( provider.getAddress ( dest ),  this,
actionDelayMillis,
                             originator )
            );
            src  =  null;
            dest  =  null;
          }
        }
        if ( src ! =  null )
        {
          println ( "Skipping last originating address \"" + src +
            "\"; no destination specified" );
        }

      }
```

```
      Enumeration e  =  actors.elements ();
      while ( e.hasMoreElements () )
      {
        Actor actor  =  (Actor) e.nextElement ();
        actor.initialize ();
      }

      Enumeration en  =  actors.elements ();
      while ( en.hasMoreElements () )
      {
        Actor actor  =  (Actor) en.nextElement ();
        actor.start ();
      }
    }
    catch ( Exception e )
    {
      println ( "Caught exception " + e );
    }
  }

  public void dispose () {
    println ( "Stopping actors" );
    Enumeration e  =  actors.elements ();
    while ( e.hasMoreElements () )
    {
      Actor actor  =  (Actor) e.nextElement ();
      actor.dispose ();
    }
  }

  public static void main ( String [] args )
  {
    if ( args.length < 6 )
    {
      System.out.println ( "Usage: makecall <server> <login> <password> <delay>
        <origin> <destination> ..." );
      System.exit ( 1 );
    }
    new makecall ( args );
  }

  public void providerChangedEvent ( ProvEv [] eventList ) {
    if ( eventList ! =  null )
    {
      for ( int i  =  0; i < eventList.length; i++ )
      {
        if ( eventList[i] instanceof ProvInServiceEv )
        {
          conditionInService.set ();
        }
      }
    }
  }
}
```

Actor.java

```
/** * Actor.java
 *
```

```
 * Copyright Cisco Systems,  Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.cisco.jtapi.extensions.*;
public abstract class Actor implements AddressObserver,  TerminalObserver,
CallControlCallObserver,  Trace
{

  public static final int ACTOR_OUT_OF_SERVICE  =  0;
  public static final int ACTOR_IN_SERVICE  = 1;
  private Tracetrace;
  protected intactionDelayMillis;
  private AddressobservedAddress;
  private Terminal observedTerminal;
  private boolean addressInService;
  private boolean terminalInService;
  protected int state  =  Actor.ACTOR_OUT_OF_SERVICE;

  public Actor ( Trace trace,  Address observed,  int actionDelayMillis ) {
    this.trace  =  trace;
    this.observedAddress  =  observed;
    this.observedTerminal  =  observed.getTerminals ()[0];
    this.actionDelayMillis  =  actionDelayMillis;
  }

  public void initialize () {

    try
    {
      if ( observedAddress ! =  null )
      {
        bufPrintln (
          "Adding Call observer to address "
          + observedAddress.getName ()
        );
        observedAddress.addCallObserver ( this );

        //Now add observer on Address and Terminal
        bufPrintln (
          "Adding Adddress Observer to address "
          + observedAddress.getName ()
        );

        observedAddress.addObserver ( this );

        bufPrintln (
          "Adding Terminal Observer to Terminal" + observedTerminal.getName ()
        );

        observedTerminal.addObserver ( this );
      }
    }
    catch ( Exception e )
    {
    }
    finally
    {
      flush ();
```

```
  }
}

public final void start () {
  onStart ();
}



public final void dispose () {

  try
  {
    onStop ();
    if ( observedAddress ! =  null )
    {

      bufPrintln (
        "Removing observer from Address "
        + observedAddress.getName ()
      );
      observedAddress.removeObserver ( this );

      bufPrintln (
        "Removing call observer from Address "
        + observedAddress.getName ()
      );
      observedAddress.removeCallObserver ( this );

    }
    if ( observedTerminal ! =  null )
    {
      bufPrintln (
        "Removing observer from terminal "
        + observedTerminal.getName ()
      );
      observedTerminal.removeObserver ( this );
    }
  }
  catch ( Exception e )
  {
    println ( "Caught exception " + e );
  }
  finally
  {
    flush ();
  }
}

public final void stop () {
  onStop ();
}


public final void callChangedEvent ( CallEv [] events ) {
  //
  // for now,  all metaevents are delivered in the
  // same package...
  //
  metaEvent ( events );
}

public void addressChangedEvent ( AddrEv [] events ) {
```

```
    for ( int i = 0; i<events.length; i++ )
    {
      Address address  =  events[i].getAddress ();
      switch ( events[i].getID () )
      {
      case CiscoAddrInServiceEv.ID:
        bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
        addressInService  =  true;
        if ( terminalInService )
        {
          if ( state ! =  Actor.ACTOR_IN_SERVICE )
          {
            state  =  Actor.ACTOR_IN_SERVICE ;
            fireStateChanged ();
          }
        }
        break;
      case CiscoAddrOutOfServiceEv.ID:
        bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
        addressInService  =  false;
        if ( state ! =  Actor.ACTOR_OUT_OF_SERVICE )
        {  // you only want to notify when you had notified earlier that you are
 IN_SERVICE
          state  =  Actor.ACTOR_OUT_OF_SERVICE;
          fireStateChanged ();
        }
        break;
      }
    }
    flush ();
  }

  public void terminalChangedEvent ( TermEv [] events ) {

    for ( int i = 0; i<events.length; i++ )
    {
      Terminal terminal  =  events[i].getTerminal ();
      switch ( events[i].getID () )
      {
      case CiscoTermInServiceEv.ID:
        bufPrintln ( "Received " + events[i] + "for " + terminal.getName ());
        terminalInService  =  true;
        if ( addressInService )
        {
          if ( state ! =  Actor.ACTOR_IN_SERVICE )
          {
            state  =  Actor.ACTOR_IN_SERVICE;
            fireStateChanged ();
          }
        }
        break;
      case CiscoTermOutOfServiceEv.ID:
        bufPrintln ( "Received " + events[i] + "for " + terminal.getName () );
        terminalInService  =  false;
        if ( state ! =  Actor.ACTOR_OUT_OF_SERVICE )
        { // you only want to notify when you had notified earlier that you are
IN_SERVICE
          state  =  Actor.ACTOR_OUT_OF_SERVICE;
          fireStateChanged ();
        }
        break;
      }
    }
    flush();
```

```
  }

  final void delay ( String action ) {
    if ( actionDelayMillis ! =  0 )
    {
      println ( "Pausing " + actionDelayMillis + " milliseconds before " + action
);
      try
      {
        Thread.sleep ( actionDelayMillis );
      }
      catch ( InterruptedException e )
      {
      }
    }
  }

  protected abstract void metaEvent ( CallEv [] events );

  protected abstract void onStart ();
  protected abstract void onStop ();
  protected abstract void fireStateChanged ();

  public final void bufPrint ( String string ) {
    trace.bufPrint ( string );
  }
  public final void bufPrintln ( String string ) {
    trace.bufPrint ( string );
    trace.bufPrint ("\n");
  }
  public final void print ( String string ) {
    trace.print ( string );
  }
  public final void print ( char character ) {
    trace.print ( character );
  }
  public final void print ( int integer ) {
    trace.print ( integer );
  }
  public final void println ( String string ) {
    trace.println ( string );
  }
  public final void println ( char character ) {
    trace.println ( character );
  }
  public final void println ( int integer ) {
    trace.println ( integer );
  }
  public final void flush () {
    trace.flush ();
  }
}
```

Originator.java

```
/** * originator.java
 *
 * Copyright Cisco Systems,  Inc.
 *
```

```
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.ms.com.*;
import com.cisco.jtapi.extensions.*;

public class Originator extends Actor
{
  AddresssrcAddress;
  StringdestAddress;
  intiteration;
  StopSignalstopSignal;
  booleanready  =  false;
  intreceiverState  =  Actor.ACTOR_OUT_OF_SERVICE;
  boolean callInIdle  =  true;

  public Originator ( Address srcAddress,  String destAddress,  Trace trace,
    int actionDelayMillis ) {
    super ( trace,  srcAddress,  actionDelayMillis );// observe srcAddress
    this.srcAddress  =  srcAddress;
    this.destAddress  =  destAddress;
    this.iteration  =  0;
  }

  protected final void metaEvent ( CallEv [] eventList ) {
    for ( int i  =  0; i < eventList.length; i++ )
    {
      try
      {
        CallEv curEv  =  eventList[i];

        if ( curEv instanceof CallCtlTermConnTalkingEv )
        {
          TerminalConnection tc  =
((CallCtlTermConnTalkingEv)curEv).getTerminalConnection ();
          Connection conn  =  tc.getConnection ();
          if ( conn.getAddress ().getName ().equals ( destAddress ) )
          {
            delay ( "disconnecting" );
            bufPrintln ( "Disconnecting Connection " + conn );
            conn.disconnect ();
          }
        }
        else if ( curEv instanceof CallCtlConnDisconnectedEv )
        {
          Connection conn  =  ((CallCtlConnDisconnectedEv)curEv).getConnection
();
          if ( conn.getAddress ().equals ( srcAddress ) )
          {
            stopSignal.canStop ();
            setCallProgressState ( true );
          }
        }
      }
      catch ( Exception e )
      {
        println ( "Caught exception " + e );
      }
      finally
      {
```

```
        flush ();
      }

    }
}

protected void makecall ()
throws ResourceUnavailableException,  InvalidStateException,
       PrivilegeViolationException,  MethodNotSupportedException,
  InvalidPartyException,  InvalidArgumentException {
  println ( "Making call #" + ++iteration + " from " + srcAddress + " to " +
    destAddress + " " + Thread.currentThread ().getName () );
  Call call  =  srcAddress.getProvider ().createCall ();
  call.connect ( srcAddress.getTerminals ()[0],  srcAddress,  destAddress );
  setCallProgressState ( false );
  println ( "Done making call" );
}


protected final void onStart () {
  stopSignal  =  new StopSignal ();
  new ActionThread ().start ();
}

protected final void fireStateChanged () {
  checkReadyState ();
}

protected final void onStop () {
  stopSignal.stop ();
  Connection[] connections  =  srcAddress.getConnections ();
  try
  {
    if ( connections ! =  null )
    {
      for (int i = 0; i< connections.length; i++ )
      {
        connections[i].disconnect ();
      }
    }
  }
  catch ( Exception e )
  {
    println (" Caught Exception " + e);
  }
}

public int getReceiverState () {
  return receiverState;
}

public void setReceiverState ( int state ) {
  if ( receiverState ! =  state )
  {
    receiverState  =  state;
    checkReadyState ();
  }
}



public synchronized void checkReadyState ()
{
  if ( receiverState  =  =  Actor.ACTOR_IN_SERVICE && state  =  =
```

```
Actor.ACTOR_IN_SERVICE )
    {
      ready  =  true;
    }
    else
    {
      ready  =  false;
    }
    notifyAll ();
  }

  public synchronized void setCallProgressState ( boolean isCallInIdle )
  {
    callInIdle  =  isCallInIdle;
    notifyAll ();
  }

  public synchronized void doAction ()
  {
    if ( !ready || !callInIdle )
    {
      try
      {
        wait ();
      }
      catch ( Exception e )
      {
        println (" Caught Exception from wait state" + e );
      }
    }
    else
    {
      if ( actionDelayMillis ! =  0 )
      {
        println ( "Pausing " + actionDelayMillis + " milliseconds before making
call " );
        flush ();
        try
        {
          wait ( actionDelayMillis );
        }
        catch ( Exception ex )
        {
        }
      }
      //make call after waking up,  recheck the flags before making the call
      if ( ready && callInIdle )
      {
        try
        {
          makecall ();
        }
        catch ( Exception e )
        {
          println (" Caught Exception in MakeCall " + e + " Thread  = " +
            Thread.currentThread ().getName ());
        }
      }
    }
  }


  class ActionThread extends Thread {
```

```
    ActionThread ( ) {
      super ( "ActionThread");
    }

    public void run () {
      while ( true )
      {
        doAction ();
      }
    }
  }

}
```

Receiver.java

```
/** * Receiver.java
 *
 * Copyright Cisco Systems,  Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

public class Receiver extends Actor
{
  Addressaddress;
  StopSignalstopSignal;
  Originatororiginator;

  public Receiver ( Address address,  Trace trace,  int actionDelayMillis,
    Originator originator ) {
    super ( trace,  address,  actionDelayMillis );
    this.address  =  address;
    this.originator  =  originator;
  }

  protected final void metaEvent ( CallEv [] eventList ) {
    for ( int i  =  0; i < eventList.length; i++ )
    {
      TerminalConnection tc  =  null;
      try
      {
        CallEv curEv  =  eventList[i];

        if ( curEv instanceof CallCtlTermConnRingingEv )
        {
          tc  =  ((CallCtlTermConnRingingEv)curEv).getTerminalConnection ();
          delay ( "answering" );
          bufPrintln ( "Answering TerminalConnection " + tc );
          tc.answer ();
          stopSignal.canStop ();
        }
      }
      catch ( Exception e )
      {
```

```
        bufPrintln ( "Caught exception " + e );
        bufPrintln ( "tc  =  " + tc );
      }
      finally
      {
        flush ();
      }
    }
  }

  protected final void onStart () {
    stopSignal  =  new StopSignal ();
  }

  protected final void onStop () {
    stopSignal.stop ();
    Connection[] connections  =  address.getConnections ();
    try
    {
      if ( connections ! =  null )
      {
        for (int i = 0; i< connections.length; i++ )
        {
          connections[i].disconnect ();
        }
      }
    }
    catch ( Exception e )
    {
      println (" Caught Exception " + e);
    }
  }

  protected final void fireStateChanged () {
    originator.setReceiverState ( state );
  }
}
```

StopSignal.java

```
/** * StopSignal.java
 *
 * Copyright Cisco Systems,  Inc.
 *
 */

class StopSignal {
  boolean stopping  =  false;
  boolean stopped  =  false;
  synchronized boolean isStopped ()
  {
    return stopped;
  }
  synchronized boolean isStopping ()
  {
    return stopping;
  }
  synchronized void stop ()
  {
```

```
    if ( !stopped )
    {
      stopping  =  true;
      try
      {
        wait ();
      }
      catch ( InterruptedException e )
      {
      }
    }
  }
  synchronized void canStop ()
  {
    if ( stopping  =  true )
    {
      stopping  =  false;
      stopped  =  true;
      notify ();
    }
  }
}
```

Trace.java

```
/** * Trace.java
 *
 * Copyright Cisco Systems,  Inc.
 *
 */
public interface Trace
{
  /**
   * bufPrint (str) puts str in buffer only.
   */
  public void bufPrint ( String string );

  /**
   * print () println () bufPrint and invoke flush ();
   */
  public void print ( String string );
  public void print ( char character );
  public void print ( int integer );
  public void println ( String string );
  public void println ( char character );
  public void println ( int integer );

  /**
   * flush out the buffer.
   */
  public void flush ();
}
```

TraceWindow.java

```
/** * TraceWindow.java
 *
 * Copyright Cisco Systems,  Inc.
 *
 */


import java.awt.*;
import java.awt.event.*;

public class TraceWindow extends Frame implements Trace
{

  TextArea textArea;
  booleantraceEnabled  =  true;
  StringBuffer buffer  =  new StringBuffer ();

  public TraceWindow (String name ) {
    super ( name );
    initWindow ();
  }

  public TraceWindow(){
    this("");
  }


  private void initWindow() {
    this.addWindowListener(new WindowAdapter () {
                              public void windowClosing(WindowEvent e){dispose
();}
                            }
                           );
    textArea  =  new TextArea();
    setSize(400, 400);
    add(textArea);
    setEnabled(true);
    this.show();

  }


  public final void bufPrint ( String str ) {
    if ( traceEnabled )
    {
      buffer.append ( str );
    }

  }

  public final void print ( String str ) {
    if ( traceEnabled )
    {
      buffer.append ( str );
      flush ();
    }
  }
  public final void print ( char character ) {
    if ( traceEnabled )
```

```
    {
      buffer.append ( character );
      flush ();
    }
  }
  public final void print ( int integer ) {
    if ( traceEnabled )
    {
      buffer.append ( integer );
      flush ();
    }
  }
  public final void println ( String str ) {
    if ( traceEnabled )
    {
      print ( str );
      print ( '\n' );
      flush ();
    }
  }
  public final void println ( char character ) {
    if ( traceEnabled )
    {
      print ( character );
      print ( '\n' );
      flush ();
    }
  }
  public final void println ( int integer ) {
    if ( traceEnabled )
    {
      print ( integer );
      print ( '\n' );
      flush ();
    }
  }

  public final void setTrace ( boolean traceEnabled ) {
    this.traceEnabled  =  traceEnabled;
  }

  public final void flush () {
    if ( traceEnabled )
    {
      textArea.append ( buffer.toString());
      buffer  =  new StringBuffer ();
    }
  }

  public final void clear () {

    textArea.setText("");
  }
}
```

Running makecall

To Invoke makecall on the client workstation, from the Windows NT command line, navigate to the **makecall** directory where JTAPI Tools directory was installed and execute the following command:

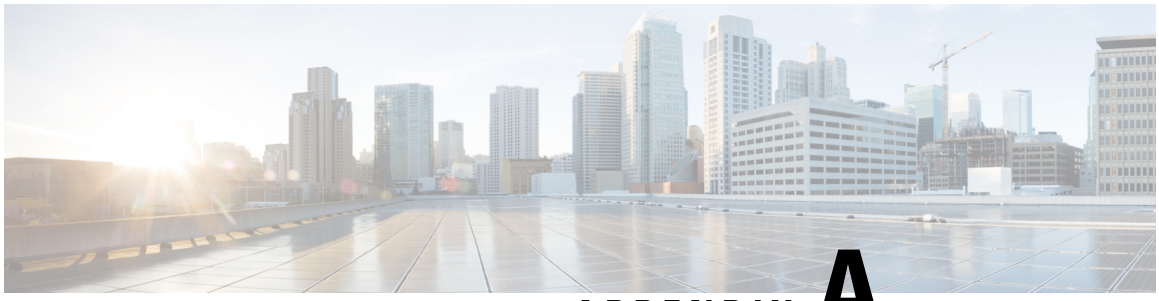
```
jview makecall <server name> <login> <password> 1000 <device 1> <device2>
```

<server name> specifies the hostname or IP address of your Cisco Unified Communications Manager.

<device1> and <device2> are directory numbers of IP phones. Make sure that the phones are part of the associated devices of a given user as administered in the Cisco Unified Communications Manager's directory administration.

<login> and <password> apply similarly as administered in the directory.

This will test that you have installed and configured everything correctly. The application will make calls between the two devices with an action delay of 1000 msecs until terminated.



APPENDIX A

Message Sequence Charts

This appendix contains message sequence charts illustrating the message flows for several scenarios.

Agent Greeting

The basic Agent Greeting use cases assume a common setup.

In the real-world scenario, an external customer calls a number and is routed through an IVR until the call is eventually offered to an agent.

IP Phones:

- Customer (1000)
- Agents (2000, 2001, 2002)
- IVRs (5000, 5001)

Scenario One

Agent Greeting Start Success

Action	Events	Call information / Notes
1. Customer dials the agent.	GC1 - CallActiveEvent GC1 - ConnCreatedEvent (1000) GC1 - ConnConnectedEvent (1000) GC1 - CallCtlConnInitiatedEv (1000) GC1 - TermConnCreatedEvent (Term of 1000) GC1 - TermConnActiveEvent (Term of 1000) GC1 - CallCtlTermConnTalkingEv (Term of 1000) GC1 - CallCtlConnDialingEv (1000) GC1 - CallCtlConnEstablishedEv (1000) GC1 - ConnCreatedEvent (2000) GC1 - ConnInprogressEvent (2000) GC1 - CallCtlConnOfferedEv (2000) GC1 - ConnAlertingEvent (2000) GC1 - CallCtlConnAlertingEv (2000) GC1 - TermConnCreatedEvent (Term of 2000) GC1 - TermConnRingingEvent (Term of 2000) GC1 - CallCtlTermConnRingingEv (Term of 2000) GC1 - ConnConnectedEvent (2000) GC1 - CallCtlConnEstablishedEv (2000) GC1 - TermConnActiveEvent (Term of 2000) GC1 - CallCtlTermConnTalkingEv (Term of 2000)	This is a basic call. Calling = 1000 (Customer) Called = 2000 (Agent)

Action	Events	Call information / Notes
2. Application gets the TerminalConnection for 2000 on GC1 and invokes addMediaStream("5000", "2000")	GC2 - CallActiveEvent GC2 - ConnCreatedEvent (5000) GC2 - ConnInprogressEvent (5000) GC2 - CallCtlConnOfferedEv (5000) GC2 - ConnAlertingEvent (5000) GC2 - CallCtlConnAlertingEv (5000) GC2 - TermConnCreatedEvent (Term of 5000) GC2 - TermConnRingingEvent (Term of 5000) GC2 - CallCtlTermConnRingingEv (Term of 5000) GC2 - ConnConnectedEvent (5000) GC2 - CallCtlConnEstablishedEv (5000) GC2 - TermConnActiveEvent (Term of 5000) GC2 - CallCtlTermConnTalkingEv (Term of 5000) GC1 - CiscoMediaStreamStartedEv (2000)	This is a server call. Calling = 2000 (Agent) Called = 5000 (IVR) The Calling Party number is as specified in the addMediaStream() method ("2000" in this case), and is available immediately from the CallActiveEvent. **Note** No connection for 2000 is created, as 2000 is "spoofed". Agent Greeting is complete.
3. Application disconnects IVR, or tester manually hangs up the IVR device.	GC2 - CallCtlTermConnDroppedEv (Term of 5000) GC2 - ConnDisconnectedEvent (5000) GC2 - CallCtlConnDisconnectedEv (5000) GC2 - CallInvalidEvent (5000) GC2 - CallObservationEndedEv GC1 - CiscoMediaStreamEndedEv (2000)	BIB call is cleaned up. Ev.isSuccessful() = true. The call continues as normal.
4. Agent finishes the conversation and ends the call	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000) GC1 - CallCtlConnDisconnectedEv (2000) GC1 - TermConnDroppedEv (Term of 1000) GC1 - CallCtlTermConnDroppedEv (Term of 1000) GC1 - ConnDisconnectedEvent (1000) GC1 - CallCtlConnDisconnectedEv (1000) GC1 - CallInvalidEvent GC1 - CallObservationEndedEv	Primary call is cleaned up.

Scenario Two

Agent Greeting Stop Success

Agent	Events	Call information / Notes
1. Customer calls the agent and the agent answers. Application invokes addMediaStream().	GC1 - CiscoMediaStreamStartedEv (2000)	Ev.getIVRCall() = Call for CG2.
2. While the greeting is played, the application invokes removeMediaStream().	GC2 - CallCtlTermConnDroppedEv (Term of 5000) GC2 - ConnDisconnectedEvent (5000) GC2 - CallCtlConnDisconnectedEv (5000) GC2 - CallInvalidEvent (5000) GC2 - CallObservationEndedEv GC1 - CiscoMediaStreamEndedEv (2000)	The Agent Greeting is cut short. The BIB call is cleaned up. Ev.isSuccessful() = true. The call continues as normal.
3. The agent finishes the conversation and ends the call.	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000) GC1 - CallCtlConnDisconnectedEv (2000) GC1 - TermConnDroppedEv (Term of 1000) GC1 - CallCtlTermConnDroppedEv (Term of 1000) GC1 - ConnDisconnectedEvent (1000) GC1 - CallCtlConnDisconnectedEv (1000) GC1 - CallInvalidEvent GC1 - CallObservationEndedEv	The primary call is cleaned up.

Scenario Three

Agent Greeting Start Failure: Resource Unavailable

Agent	Event	Call information / Notes
1. Customer dials the Agent	GC1 - CallActiveEvent GC1 - ConnCreatedEvent (1000) GC1 - ConnConnectedEvent (1000) GC1 - CallCtlConnInitiatedEv (1000) GC1 - TermConnCreatedEvent (Term of 1000) GC1 - TermConnActiveEvent (Term of 1000) GC1 - CallCtlTermConnTalkingEv (Term of 1000) GC1 - CallCtlConnDialingEv (1000) GC1 - CallCtlConnEstablishedEv (1000) GC1 - ConnCreatedEvent (2000) GC1 - ConnInprogressEvent (2000) GC1 - CallCtlConnOfferedEv (2000) GC1 - ConnAlertingEvent (2000) GC1 - CallCtlConnAlertingEv (2000) GC1 - TermConnCreatedEvent (Term of 2000) GC1 - TermConnRingingEvent (Term of 2000) GC1 - CallCtlTermConnRingingEv (Term of 2000) GC1 - ConnConnectedEvent (2000) GC1 - CallCtlConnEstablishedEv (2000) GC1 - TermConnActiveEvent (Term of 2000) GC1 - CallCtlTermConnTalkingEv (Term of 2000)	This is a basic call Calling = 1000 (Customer) Called = 2000 (Agent)
2. The application gets the TerminalConnection for 2000 on GC1and invokes addMediaStream("5000", "2000").		No BIB call is created. JTAPI throws a ResourceUnavailableException with text as "Unable to allocate built in bridge resource". The call continues as normal.
3. The agent finishes the conversation and ends the call.	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000)	The primary call is cleaned up.

API for Exposing Built-in-Bridge Status

Phone TermA, CTI port TermB, and RoutePoint TermC are in application’s control list.

Use Case One

BIB is disabled on service parameters and device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	False	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Two

BIB is disabled on service parameters page and enabled on device page of TermA..

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	True	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Three

BIB is enabled on service parameters page and disabled on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	False	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Four

BIB is enabled on service parameters page and set to default on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	True	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Five

Phone TermA is not registered. BIB is enabled on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	InvalidStateException	
Add observers and register TermB and TermC.		
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Backward Compatibility Enhancements

This feature is not expected to change the performance or scalability of Cisco Unified Communications Manager JTAPI. There is no change in the number of events between JTAPI and CTI. For features involving GCID changes this feature introduces one extra event which should not cause any performance issues.

In all cases events listed below are delivered to call observers when only one party is in control list. TERMA indicates terminal of A.

Scenario One

A calls B, B transfers the call to C. GC1 is the call between A and B, GC2 is the consult call between B and C. Similar events are delivered for Conference and other features.

Action	Events
B completes the transfer. Events to call observer on C	GC2 CiscoTransferStartEv Cause: CAUSE_NORMAL Reason = REASON_TRANSFER CallActiveEv GC1 Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER ConnCreatedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER CiscoCallChangedEv SurvingCall = GC1, original call = GC2 CiscoCause: NORMAL Reason: REASON_TRANSFER

Action	Events
Events delivered to CallObserver of B (transfer controller)	

Action	Events
	ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER TermConnCreatedEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER TermConnActiveEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER CallCtlTermConnTalkingEv TERM C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER ConnConnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER GC2: ConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: ConnDisconnectedEv C REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: TermConnDropped TERMB REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: CalInvalid REASON = REASON_TRANSFER Cause:CAUSE_NORMAL GC1: CallCtlTermConnHeldEv TERMB REASON = REASON_TRANSFER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Events
	GC2: ConsultCallActive REASON = NORMAL Cause:CAUSE_NEW_CALL GC2: ConnCreatedEv B REASON = NORMAL Cause:CAUSE_NORMAL GC2: ConnConnectedEv B REASON = NORMAL Cause:CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN GC1: CallCtlConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC1: TermConnDroppedEv TERMB REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN GC1: CallCtlTermConnDroppedEv TERMB REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: ConnDisconnectedEv A REASON = REASON_TRANSFER GC1: CallCtlConnDisconnectedEv A REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: CallInvalidEv REASON = REASON_TRANSFER GC2: ConnDisconnectedEv C REASON = REASON_TRANSFER GC2: CallCtlConnDisconnectedEv C REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: TermConnDroppedEv TERMB REASON = REASON_TRANSFER GC2: CallCtlTermConnDroppedEv TERMB REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER

Action	Events
	GC2: ConnDisconnectedEv B REASON = REASON_TRANSFER GC2: CallCtlConnDisconnectedEv B REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: CallInvalidEv REASON = REASON_TRANSFER GC2: CallObservationEndedEv REASON = NORMAL Cause:CAUSE_NORMAL GC1 CiscoTransferEndEv REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2 CallObservationEndedEv REASON = NORMAL Cause:CAUSE_NORMAL

Scenario Two

A calls B, call = GC1. B parks the call at 99999. C unparks the call using call GC2.

Action	Events
Events delivered to call observer on A when call is parked. When call is unparked using GC2	

Action	Events
	GC1: ConnDisconnectedEv B REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnCreatedEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: ConnInProgressEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnQueuedEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: CiscoCallChangedEv Surviving = GC2 origcall = GC1 address = A REASON = REASON_UNPARK CallActiveEv REASON = REASON_UNPARK Cause: CAUSE_NEW_CALL GC2: ConnCreatedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: TermConnCreatedEv TERMA REASON = REASON_UNPARK GC2: TermConnActiveEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL

Action	Events
	CallControlCause: CAUSE_PARK GC1: ConnDisconnectedEv 9999 REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv 9995 REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: TermConnDroppedEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnDisconnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: CallInvalidEv REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallObservationEndedEv REASON = NORMAL Cause: CAUSE_NORMAL GC2: ConnCreatedEv C REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv C REASON = UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv C REASON = UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK

Scenario Three

A calls B, B has forward no answer to C. B does not answer and call is offered to C.

Action	Events
Events delivered to call observer on A.	

Action	Events
	GC1: CallActiveEv REASON = NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnInitiatedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMA REASON = NORMAL GC1: TermConnActiveEv TERMA REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnTalkingEv TERMA REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnDialingEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv

Action	Events
	REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: ConnInProgressEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnAlertingEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnConnectedEv C REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Scenario Four

A calls B, B redirects the call to C.

Action	Events
Events delivered to call observer on B.	

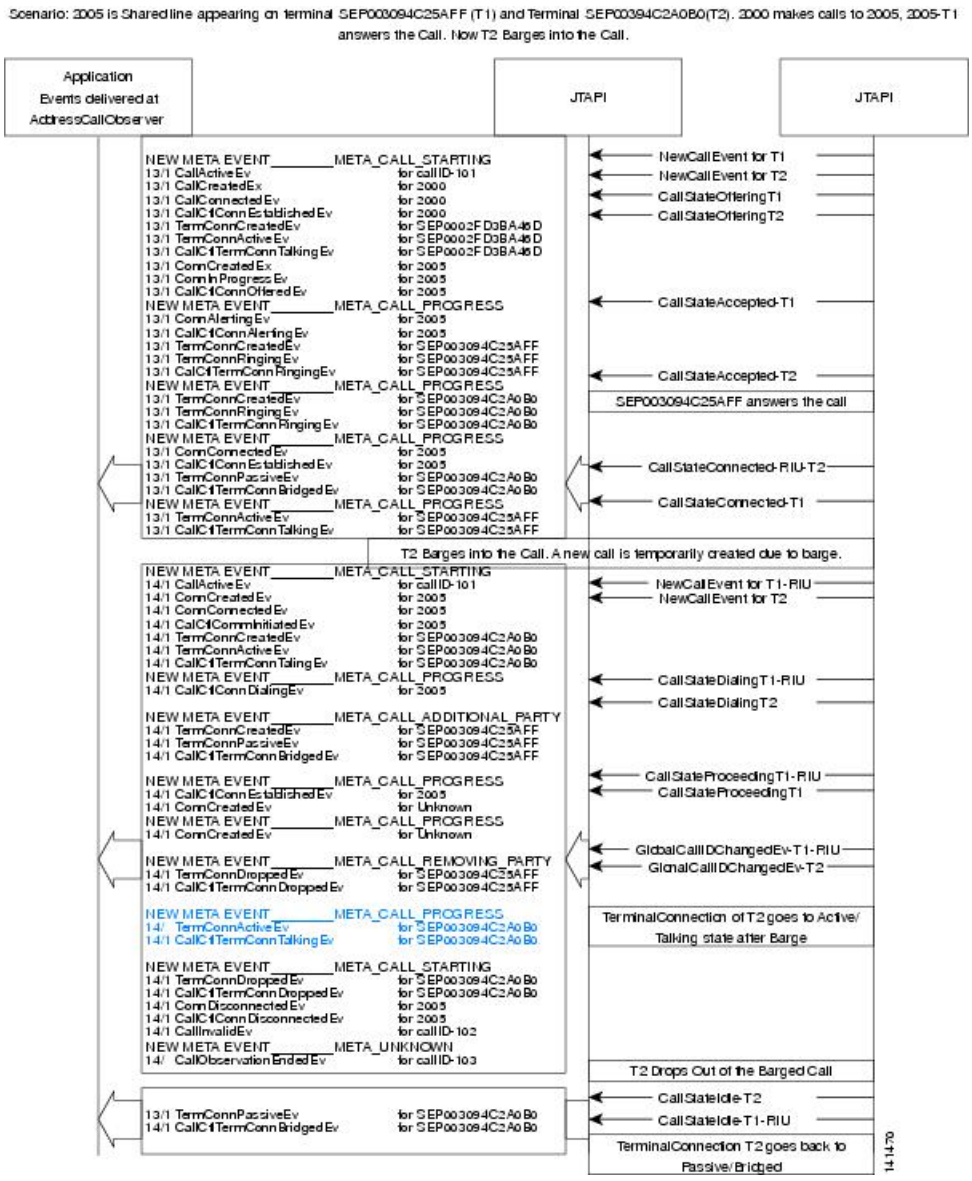
Action	Events
	GC1: CallActiveEv REASON = NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMB REASON = NORMAL Cause: Other: 0 GC1: TermConnRingingEv TERMB REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEvImpl TERMB REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnDisconnectedEv A

Action	Events
	REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: TermConnDroppedEv TERMB REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMB REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnDisconnectedEv B REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: CallInvalidEv REASON = REDIRECT Cause: CAUSE_NORMAL

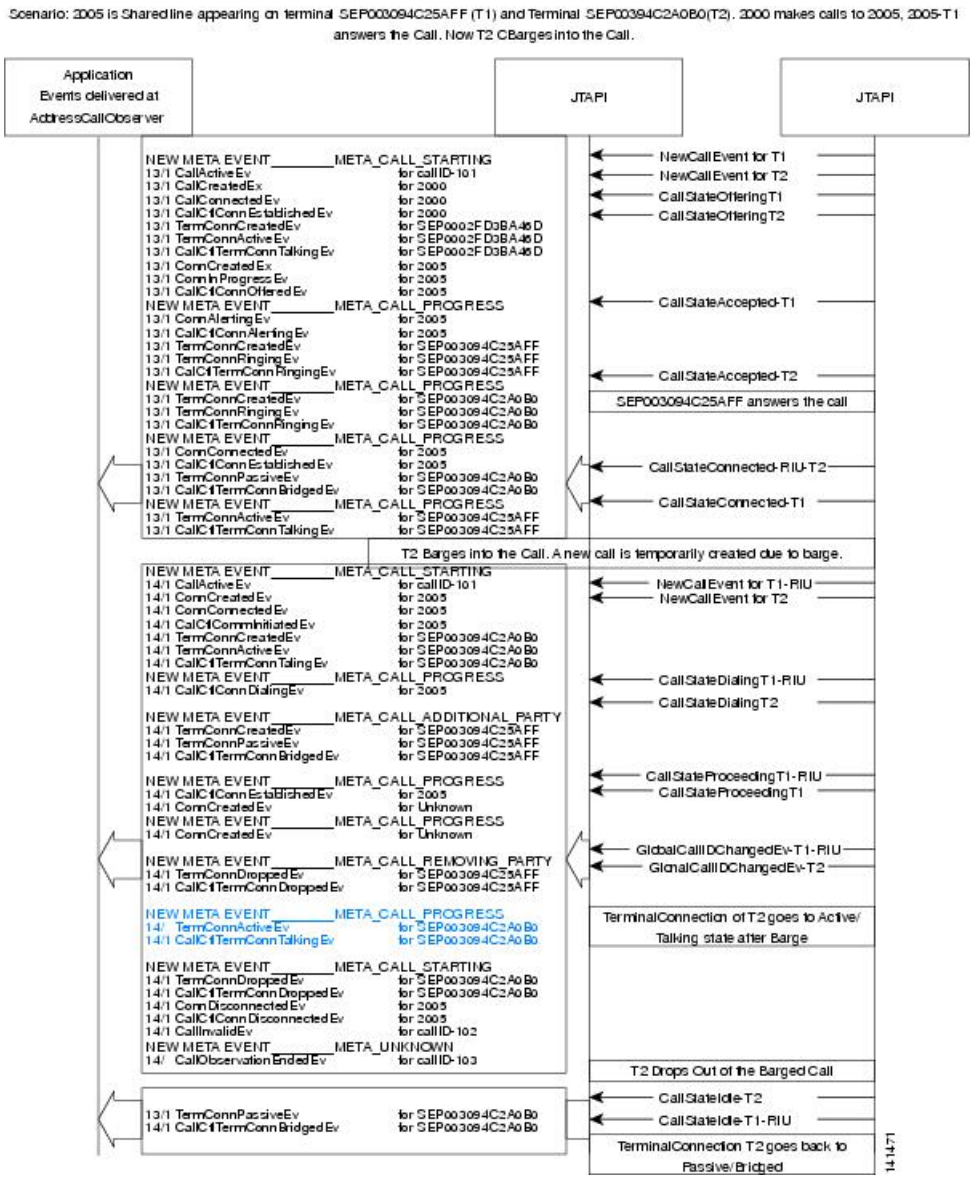
Barge and Privacy

The following diagrams illustrate the message flows for Barge and Privacy.

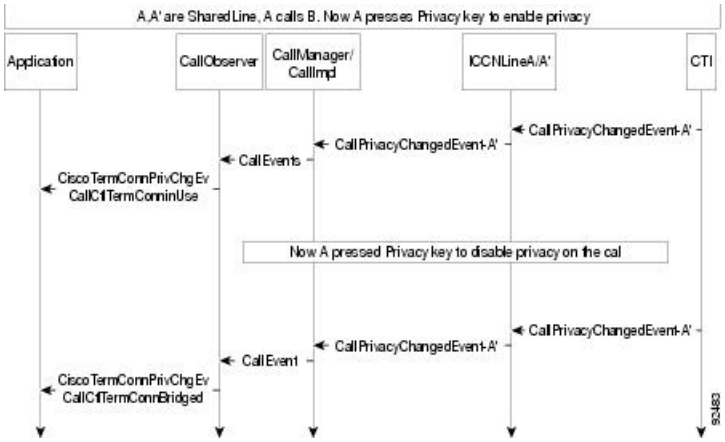
Barge


Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005, 2005-T1 answers the Call. Now T2 Barges into the Call.
Application
Events delivered at
AddressCallObserver
JTAPI
JTAPI
NEW META EVENT META_CALL_STARTING
13/1 CallActiveEv for callID-101
13/1 CallCreatedEx for 2000
13/1 CallConnectedEv for 2000
13/1 CallCtlConnEstablishedEv for 2000
13/1 TermConnCreatedEv for SEP0002FD3BA46D
13/1 TermConnActiveEv for SEP0002FD3BA46D
13/1 CallCtlTermConnTalkingEv for SEP0002FD3BA46D
13/1 ConnCreatedEx for 2005
13/1 ConnInProgressEv for 2005
13/1 CallCtlConnOfferedEv for 2005
NEW META EVENT META_CALL_PROGRESS
13/1 ConnAlertingEv for 2005
13/1 CallCtlConnAlertingEv for 2005
13/1 TermConnCreatedEv for SEP003094C25AFF
13/1 TermConnRingingEv for SEP003094C25AFF
13/1 CallCtlTermConnRingingEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnCreatedEv for SEP003094C2A0B0
13/1 TermConnRingingEv for SEP003094C2A0B0
13/1 CallCtlTermConnRingingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 ConnConnectedEv for 2005
13/1 CallCtlConnEstablishedEv for 2005
13/1 TermConnPassiveEv for SEP003094C2A0B0
13/1 CallCtlTermConnBridgedEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnActiveEv for SEP003094C25AFF
13/1 CallCtlTermConnTalkingEv for SEP003094C25AFF
NEW META EVENT META_CALL_STARTING
14/1 CallActiveEv for callID-101
14/1 ConnCreatedEv for 2005
14/1 ConnConnectedEv for 2005
14/1 CallCtlConnInitiatedEv for 2005
14/1 TermConnCreatedEv for SEP003094C2A0B0
14/1 TermConnActiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnTalingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
14/1 CallCtlConnDialingEv for 2005
NEW META EVENT META_CALL_ADDITIONAL_PARTY
14/1 TermConnCreatedEv for SEP003094C25AFF
14/1 TermConnPassiveEv for SEP003094C25AFF
14/1 CallCtlTermConnBridgedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 CallCtlConnEstablishedEv for 2005
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_PROGRESS
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_REMOVING_PARTY
14/1 TermConnDroppedEv for SEP003094C25AFF
14/1 CallCtlTermConnDroppedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 TermConnActiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnTalkingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_STARTING
14/1 TermConnDroppedEv for SEP003094C2A0B0
14/1 CallCtlTermConnDroppedEv for SEP003094C2A0B0
14/1 ConnDisconnectedEv for 2005
14/1 CallCtlConnDisconnectedEv for 2005
14/1 CallInvalidEv for callID-102
NEW META EVENT META_UNKNOWN
14/1 CallObservationEndedEv for callID-103
13/1 TermConnPassiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnBridgedEv for SEP003094C2A0B0
NewCallEvent for T1
NewCallEvent for T2
CallStateOfferingT1
CallStateOfferingT2
CallStateAccepted-T1
CallStateAccepted-T2
SEP003094C25AFF answers the call
CallStateConnected-RIU-T2
CallStateConnected-T1
T2 Barges into the Call. A new call is temporarily created due to barge.
NewCallEvent for T1-RIU
NewCallEvent for T2
CallStateDialingT1-RIU
CallStateDialingT2
CallStateProceedingT1-RIU
CallStateProceedingT1
GlobalCallIDChangedEv-T1-RIU
GlobalCallIDChangedEv-T2
TerminalConnection of T2 goes to Active/ Talking state after Barge
T2 Drops Out of the Barged Call
CallStateIdle-T2
CallStateIdle-T1-RIU
TerminalConnection T2 goes back to Passive/Bridged
141470

CBarge


Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005, 2005-T1 answers the Call. Now T2 CBarges into the Call.
Application Events delivered at AddressCallObserver
JTAPI
JTAPI
NEW META EVENT META_CALL_STARTING
13/1 CallActiveEv for callID-101
13/1 CallCreatedEx for 2000
13/1 CallConnectedEv for 2000
13/1 CallC1ConnEstablishedEv for 2000
13/1 TermConnCreatedEv for SEP0002FD3BA46D
13/1 TermConnActiveEv for SEP0002FD3BA46D
13/1 CallC1TermConnTalkingEv for SEP0002FD3BA46D
13/1 ConnCreatedEx for 2005
13/1 ConnInProgressEv for 2005
13/1 CallC1ConnOfferedEv for 2005
NEW META EVENT META_CALL_PROGRESS
13/1 ConnAlertingEv for 2005
13/1 CallC1ConnAlertingEv for 2005
13/1 TermConnCreatedEv for SEP003094C25AFF
13/1 TermConnRingingEv for SEP003094C25AFF
13/1 CalC1TermConnRingingEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnCreatedEv for SEP003094C2A0Bo
13/1 TermConnRingingEv for SEP003094C2A0Bo
13/1 CallC1TermConnRingingEv for SEP003094C2A0Bo
NEW META EVENT META_CALL_PROGRESS
13/1 ConnConnectedEv for 2005
13/1 CallC1ConnEstablishedEv for 2005
13/1 TermConnPassiveEv for SEP003094C2A0Bo
13/1 CallC1TermConnBridgedEv for SEP003094C2A0Bo
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnActiveEv for SEP003094C25AFF
13/1 CallC1TermConnTalkingEv for SEP003094C25AFF
NEW META EVENT META_CALL_STARTING
14/1 CallActiveEv for callID-101
14/1 ConnCreatedEv for 2005
14/1 ConnConnectedEv for 2005
14/1 CalC1CommInitiatedEv for 2005
14/1 TermConnCreatedEv for SEP003094C2A0Bo
14/1 TermConnActiveEv for SEP003094C2A0Bo
14/1 CallC1TermConnTalingEv for SEP003094C2A0Bo
NEW META EVENT META_CALL_PROGRESS
14/1 CallC1ConnDialingEv for 2005
NEW META EVENT META_CALL_ADDITIONAL_PARTY
14/1 TermConnCreatedEv for SEP003094C25AFF
14/1 TermConnPassiveEv for SEP003094C25AFF
14/1 CallC1TermConnBridgedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 CallC1ConnEstablishedEv for 2005
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_PROGRESS
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_REMOVING_PARTY
14/1 TermConnDroppedEv for SEP003094C25AFF
14/1 CallC1TermConnDroppedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/ TermConnActiveEv for SEP003094C2A0Bo
14/1 CallC1TermConnTalkingEv for SEP003094C2A0Bo
NEW META EVENT META_CALL_STARTING
14/1 TermConnDroppedEv for SEP003094C2A0Bo
14/1 CallC1TermConnDroppedEv for SEP003094C2A0Bo
14/1 ConnDisconnectedEv for 2005
14/1 CallC1ConnDisconnectedEv for 2005
14/1 CallInvalidEv for callID-102
NEW META EVENT META_UNKNOWN
14/ CallObservationEndedEv for callID-103
13/1 TermConnPassiveEv for SEP003094C2A0Bo
14/1 CallC1TermConnBridgedEv for SEP003094C2A0Bo
NewCallEvent for T1
NewCallEvent for T2
CallStateOfferingT1
CallStateOfferingT2
CallStateAccepted-T1
CallStateAccepted-T2
SEP003094C25AFF answers the call
CallStateConnected-RIU-T2
CallStateConnected-T1
T2 Barges into the Call. A new call is temporarily created due to barge.
NewCallEvent for T1-RIU
NewCallEvent for T2
CallStateDialingT1-RIU
CallStateDialingT2
CallStateProceedingT1-RIU
CallStateProceedingT1
GlobalCallIDChangedEv-T1-RIU
GlobalCallIDChangedEv-T2
TerminalConnection of T2 goes to Active/ Talking state after Barge
T2 Drops Out of the Barged Call
CallStateIdle-T2
CallStateIdle-T1-RIU
TerminalConnection T2 goes back to Passive/Bridged
141471

Privacy



Call Control Discovery

Scenario 1: A Calls 1000 in Other Cluster (SAF ICT)

Action	Result	Call info
A dials 1000, this call is first be intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnDialingEv - ATermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv - TA	
Called Party is 1000	ConnCreatedEv 1000 ConnInProgressEv 1000 CallCtlConnOfferedEv 1000	getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = 1000

Scenario 2: A Calls B, Within Same Cluster. B Redirects the Call to 1000, Which Is in Another Cluster (SAF ICT)

Action	Result	Call info
A calls B within the same cluster	CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEvImpl TB	
B redirects the call to 1000	TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv 1000	getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = B

Scenario 3: A Calls 1000 Which Is in the Other Cluster (SAF ICT Bandwidth Is Low)

Action	Result	Call info
A dials 1000, this call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnDialingEv - ATermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv - TA	

Action	Result	Call info
SIP trunk rejects this as bandwidth is not available	CallCtlConnEstablishedEv –A ConnCreatedEv 1000 ConnConnectedEv 1000 CallCtlConnOfferedEv 1000	getCallingAddress() = A getCalledAddress() = 1000 getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getLastRedirectedAddress() = “”
CCD Requesting feature starts PSTN failover by directing this caller to 1000’s PSTN failover number. Call is sent out to a PSTN gateway, and calling side moves to Ringback state.	CallCtlConnNetworkReachedEv 1000 CallCtlConnNetworkAlertingEv 1000	CiscoFeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCallingAddress() = A getCalledAddress() = 1000 getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getLastRedirectedAddress() = 1000

Scenario 4: A Calls B Within the Cluster. B Redirects the Call to 1000 (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B within the same cluster	CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEvImpl TB	

Action	Result	Call info
B redirects the call to 1000. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number. Call is sent out to a PSTN gateway	ConnCreatedEv 1000 ConnConnectedEv 1000	getCallingAddress() = A getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000 Reason = REASON_SAF_CCD_PSTN_FAILOVER

Scenario 5: A Calls B, B Transfers the Call to 1000 (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB	

Action	Result	Call info
B makes a consult call to 1000. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk.	GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 CallCtlConnEstablishedEv B GC2 ConnCreatedEv 1000 GC2 ConnCreatedEv 1000 GC2 CallCtlConnOfferedEv 1000	
SIP trunk rejects this call as bandwidth is not available CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number (or as configured on the server). Call is sent out to a PSTN gateway.	GC2 CallCtlConnNetworkReachedEv 1000 GC2 CallCtlConnNetworkAlertingEv 1000	CiscofeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000

Action	Result	Call info
B completes the transfer	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoTransferStartedEv GC2 CiscoCallChangedEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv 1000 GC1 ConnAlertingEv 1000 GC1 CallCtlConnAlertingEv 1000 GC1 TermConnCreatedEv 1000 GC1 TermConnRingingEv 1000 GC1 CallCtlTermConnRingingEvImpl 1000 GC2 TermConnDroppedEv 1000 GC2 CallCtlTermConnDroppedEv 1000 GC2 ConnDisconnectedEv1408972 1000 GC2 CallCtlConnDisconnectedEv 1000 GC1 TermConnDroppedEv B GC1 CallCtlTermConnDroppedEv B GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectecEv B GC2 TermConnDroppedEv B GC2 CallCtlTermConnDroppedEv B GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectecEv B GC2 CallInvalidEv GC1 CiscoTransferEndEv	Reason = REASON_TRANSFEREDCALL
A and 1000 come in direct call	GC1 ConnConnectedEv 1000 GC1 CallCtlConnEstablishedEv 1000 GC1 termConnActiveEv 1000 GC1 CallCtlTermConnTalkingEv 1000	

Scenario 6: A Calls B, B Consults 1000 and Adds It to Conference (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB	
B makes a consult call to 1000 for conference. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk.	GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 CallCtlConnEstablishedEv B GC2 ConnCreatedEv 1000 GC2 ConnCreatedEv 1000 GC2 CallCtlConnOfferedEv 1000	

Action	Result	Call info
SIP trunk rejects this call as no more bandwidth is available CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number; call is sent out to a PSTN gateway.	GC2 CallCtlConnNetworkReachedEv 1000 GC2 CallCtlConnNetworkAlertingEv 1000	CiscofeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000
B completes the conference	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoConferenceStartedEv GC2 termConnDroppedEv B GC2 CallCtlTermConnDroppedEv B Gc2 ConnDisconnectedEv B GC2 CallCtlConnDisConnectedEv B GC1 CallCtlTermConnTalkingEv B GC2 CiscoCallChangedEv GC1 ConnCreatedEv 1000 GC1 ConConnectedEv 1000 GC1 CallCtlConnEstablishedEv 1000 GC1 TermConnCreatedEv 1000 GC1 TermConnActiveEv 1000 GC1 CallCtlTermConnTalkingEv 1000 GC2 TermConnDroppedEv 1000 GC2 CallCtlTermConnDroppedEv 1000 GC2 ConnDisconnectedEv 1000 GC2 CallCtlConnDisconnectedEv 1000 GC2 CallInvalidEv GC1 CiscoTermConnSelectChangedEv B GC1 CiscoTermConnSelectChangedEv B	Reason = REASON_CONFERENCE

CallFwdAll Keys Press Notification

(Scenario 1): Application Is Observing A; A Goes Off-Hook

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	
A goes off-hook.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

(Scenario 2): A Goes Off-Hook; Application Starts Observing A

Action	Result	Call info
A goes off-hook	No Event is delivered	
Application starts observing A	CiscoAddrInServiceEv – A GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_SNAPSHOT

(Scenario 3): Application Is Observing A; User Presses CFwdAll Soft Key on Phone A in On-Hook State

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	

Action	Result	Call info
User presses CFwdAll soft key on phone A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

(Scenario 4): User Presses CFwdAll Soft Key on Phone A Goes in On-Hook State; Application Starts Observing A

Action	Result	Call info
User presses CFwdAll soft key on phone A	No event is delivered	
Application starts observing A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_SNAPSHOT

(Scenario 5): Application Is Observing A; A Goes Off-Hook and Presses CFwdAll Soft Key

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	

Action	Result	Call info
A goes off-hook.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall_CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
A presses CFwdAll soft key	No Event is delivered	

(Scenario 6): Application Is Observing A; User Presses CFwdAll Key on Phone A and Dial 9999(B) to Set the CFA Destination as B; User Then Presses CFwdAll Soft Key Again to Cancel the CallFwdAll

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	
User presses CFwdAll soft key on phone A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv – A TermConnCreatedEv – TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
User dials B to set CFA destination as B	GC1: CallCtlConnDialingEv – A CallCtlConnEstablishedEv – A TermConnDroppedEv – TA CallCtlTermConnDroppedEv – TA ConnDisconnectedEv – A CallCtlConnDisconnectedEv – A CallInvalidEv	currentCalling = A currentCalled = null currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Action	Result	Call info
User presses CFwdAll soft key on phone A to cancel CFA	GC2: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA TermConnDroppedEv – TA CallCtlTermConnDroppedEv – TA ConnDisconnectedEv – A CallCtlConnDisconnectedEv – A CallInvalidEv	(GC2)TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_CLEAR currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

Call Recording for SIP or TLS Authenticated calls

Scenario One

Recording behavior for an authenticated Phone when Service Parameter **Authenticated Phone Recording** set to **Do not Allow Recording**.

B is an Authenticated Phone having selective recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

Action	Events	Call information
termConnB.startRecording()	Recording fails with PlatformException	PlatformException.getErrorCode= CiscoJtapiException.CTIERR_SECURITY_CAPABILITY_MISMATCH

Scenario Two

Recording behavior for an authenticated Phone when Service Parameter **Authenticated Phone Recording** set to **Allow Recording**.

B is an Authenticated Phone having selective recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

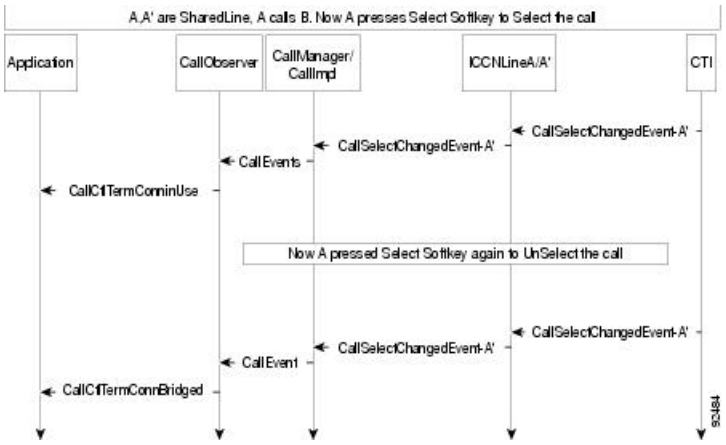
Action	Events	Call information
termConnB.startRecording()	Along with the regular events for call answer, the following events will also be delivered to the call observer: CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL CiscoTermConnRecordingTargetInfoEv CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv TA Cause: CAUSE_NORMAL	Calling: A Called: B

B is an Authenticated Phone having auto recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

Action	Events	Call Information
When B answers	Along with the regular events for call answer, the following events will also be delivered to the call observer: CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL CiscoTermConnRecordingTargetInfoEv CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv TA Cause: CAUSE_NORMAL	Calling: A Called: B

CallSelect and UnSelect

The following diagram illustrates the message flows for CallSelect and UnSelect.



Cius Persistency

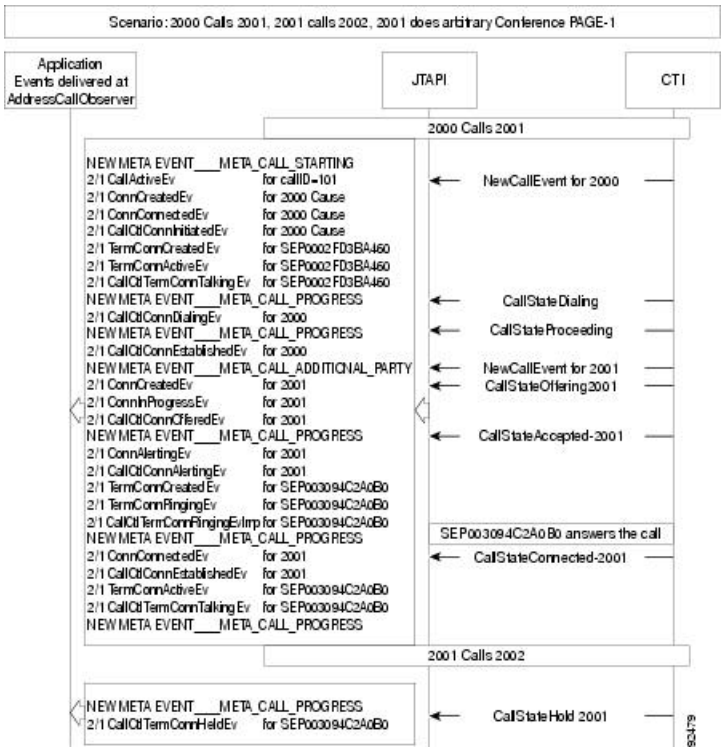
Use Cases for Cius Persistency

Usecase	Events on Provider observer	Info
Application has a wireless device TermA in its control list which is registered with IPv4 address 1.1.1.1	ProvInServiceEv	((CiscoTerminal)(Provider.getTerminal(TermA))).getIPV4Address() = 1.1.1.1
The device moves from one WiFi N/W to another resulting in the change in the IPv4 address from 1.1.1.1 to 2.2.2.2	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV4 Ev.getIPV4Address() = 2.2.2.2 ((CiscoTerminal)(Ev.getTerminal()).getIP4Address() = 2.2.2.2
The deivce moves from a IPv4 n/w to a Ipv6 n/w With new ip as 1::1	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV6 Ev.getIPV6Address() = 1::1 ((CiscoTerminal)(Ev.getTerminal()).getIP6Address() = 1::1
The Device is docked on a base station connected to the ethernet resulting in a change in IP address to 3.3.3.3	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV4 Ev.getIPV4Address() = 3.3.3.3 Ev.getTerminal() = TermA ((CiscoTerminal)(Ev.getTerminal()).getIP4Address() = 3.3.3.3

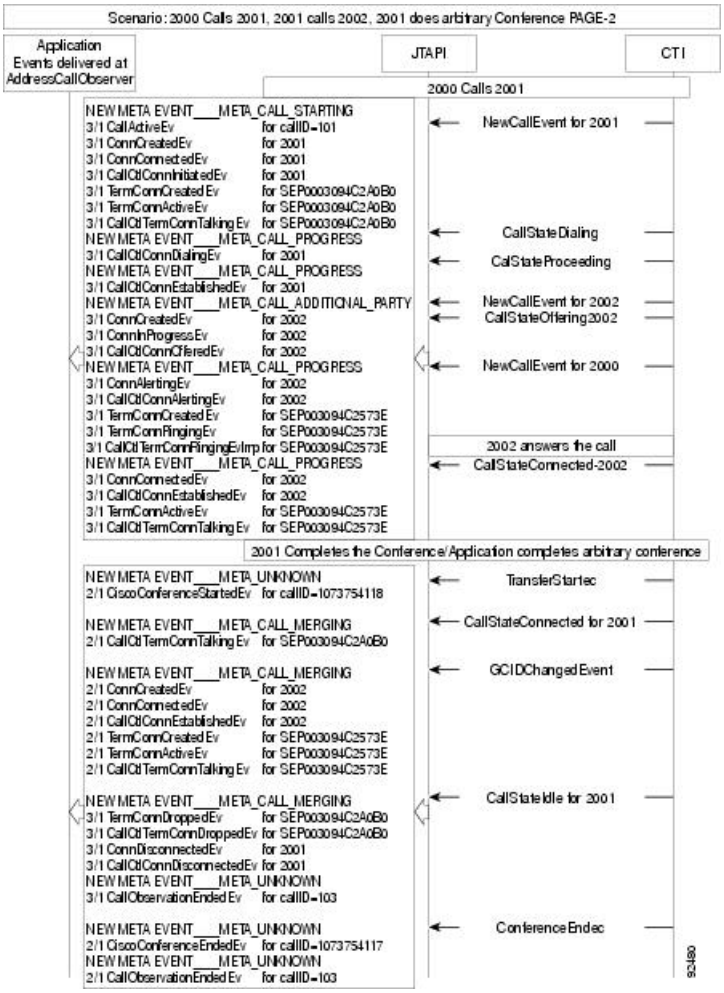
Conference and Join

The following diagrams illustrate the message flows for Conference and Join.

Join/Arbitrary Conference



Join/Arbitrary Conference—Page 2



Consult Conference

The message flow for Consult Conference acts the same as the flow for Arbitrary Conference.

Join Across Lines with Enhancements

The message flows for Join Across Lines with Enhancements are described in following tables. A, C, D, E and F are addresses on different terminals. B1 and B2 are addresses on the same terminal, TermB.

Action	Events
Application conferences the two calls on B1 and B2 by invoking GC1.conference(GC2) to chain two conference calls.	**Events to CallObserver of A, C and B1:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1
	Event for CallObserver at B2, D & E: ConnDisconnectedEv B2 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection of Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 & Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 & GC2

Action	Events
Application invokes GC2.conference (GC1) to chain two conference calls.	**Event for CallObserver at B2, D & E:** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2
	Events for CallObservers at A, B1 & C: ConnDisconnectedEv B1 GC1 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1

Action	Events
A, B1, C are in conference-1 (GC1), B1, D, E are in conference-2 (GC2), B2, F, G are in conference-3 (GC-3) Application completes conference at C by initiating GC1.conference(GC2, GC3) setting B1 as controller.	**Event for CallObserver at A, B1 & C:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC1 Ev.getAddedConnection will return connection for Conference-3 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 & Conference-3 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 & GC3

Action	Events
	Event for CallObserver at B1, D & E: ConnDisconnectedEv B1 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1-GC2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2
	Event for CallObserver at B2, F & G: ConnDisconnectedEv B2 GC3 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC3 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC3

Action	Events
Application sets the requestor as B2 and calls GC2.conference(GC1) getControllerAddress() returns B2. getOriginalControllerAddress() returns B1.	A CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
If application uses B1 as request controller in the above setup getControllerAddress() returns B1. getOriginalControllerAddress() returns B1.	Events are same as above

CTI Manager Redundancy Handling with Least Priority CTIManager Configured

Identify a CTIManager as least priority:

Application can mark one of the CTIManagers in the initial CTIManager redundancy group or configure a new one (not part of the initial group) by invoking setLeastPriorityCtiServer().

CTI Manager Redundancy Handling with Least Priority CTI Server Set

Scenario 1: Set least priority without specifying fallback Initiation time

1. Start application and set a CTIManager as least priority. Assume CTIManager redunancy list is CT1,CTI2,CTI3.
2. Application loses connectivity to CTI1.
3. Application loses connectivity to CTI3.
4. CTI1 is reachable now.
5. Fallback is started 5 min from now if a CTI server is reachable post it.
6. Post 5 min, CTI1 is still reachable.

Action	Events
Application invokes CiscoProvider.setLeastPriorityCtiServer(CTI2).	
Application connects to CTI3.	
Application connects to CTI2.	CiscoProvConnToLeastPriorCtiServerEv
JTAPI is able to identify CTI1 reachability.	CiscoProvPrimNwReachableEv
JTAPI initiates application fallback to CTI1	Once connected to CTI1, JTAPI delivers CiscoProvFallbackToPrimNwCompltdEv event

Scenario 2: Application initiates a forced fallback

1. Start application and set a CTIManager as least priority. Assume CTIManager redunancy list is CT1,CTI2,CTI3.
2. Application loses connectivity to CTI1.
3. Application loses connectivity to CTI3.
4. CTI1 is reachable now.
5. Application monitors if CTI2 is reachable now.

Action	Events	Result
Application invokes CiscoProvider.setLeastPriorityCtiServer(CTI2,600) where 600 is the fallback initiation time.		
Application connects to CTI3.		

Action	Events	Result
Application connects to CTI2.	CiscoProvConnToLeastPriorCtiServerEv	
JTAPI is able to identify CTI1 reachability.	CiscoProvPrimNwReachableEv delivered Application queries CiscoProvPrimNwReachableEv. getReachableCtiServers() returns CTI1	
Application invokes CiscoProvider.isCtiServerAvailable(CTI2)		JTAPI return true if CTI2 was reachable now.
Application invokes CiscoProvider.initiateFallback(CTI2)		JTAPI initiates fallback to CTI2 if reachable. CiscoProvFallbackTo PrimNwCompltdEv event is returned if fallback was successful.

CTI Remote Device

Use Cases

- Group 1: Get/Add/Remove/Update on Remote Destinations
- Group 2: CTIRD Incoming/Outgoing/Disconnect/Redirect/Hold/Resume and shared-line call scenarios)
- Group 3 (CUCSF registration and unregistration, for Normal SIP mode <-> Extend mode, and terminal switching scenarios
- Group 4: Set/Reset Active Remote Destination scenarios
- Group 5: CTIRD Transfer/Conference/Multiple-Calls call scenarios
- Group 6: CTIRD URI-Dialing basic Incoming & Outgoing DVO call scenarios

CTI Remote Device Use Cases Group 1

Scenario 1-1 (Expose All RDs Information on a CTI Remote Device to Application)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermA.		TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Use Cases Group 1: Get/Add/Remove/Update on Remote Destinations

Pre-conditions on Use Cases group 1 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "CTIRD-A", Line A (DN: 1000))
- Remote Destination 1 (Name: "RD1-A", Number: "4081001111", Active RD: true)
- Remote Destination 2 (Name: "RD2-A", Number: "4081002222", Active RD: false)
- Device B (IP Phone - Name: "SEP000DED47D023", Line B (DN: 2000)
- Device C (CTI Remote Device - Name: "CTIRD-C", Line C (DN: 3000))
- No Remote Destination configured.

Scenario 1-2 (Expose Active RDs Information on a CTI Remote Device to Application)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.getActiveRemoteDestinations() on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getActiveRemoteDestinations() on TermA.		TermA.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true

Scenario 1-3 (Fetch RD Information on a CTI Remote Device That Has No RD Configured)

User1 has "CTI Remote Device C" in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal C.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermC.		TermC.getAllRemoteDestinations() = null.

Scenario 1-4 (Fetch RD Information on a 'Non-CTI Remote Device')

User1 has "Device B" IP Phone in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal B.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermB.		TermB.getAllRemoteDestinations() = null.

Scenario 1-5 (Fetch Active RD Information on a CTI Remote Device That Has No Active RD Configured)

User1 has "CTI Remote Device C" in the control list. User invokes CiscoRemoteTerminal.getActiveRemoteDestinations() on terminal C.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermC.		TermC.getActiveRemoteDestinations() = null.

Scenario 1-6 (Set a Non-Active RD as a New Active RD on a 'CTI Remote Device', Where There Is Already an Existing Active RD for This Device)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.setActiveRemoteDestination("4081002222", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("4081002222", true) on TermA.	CiscoProvTerminal RemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminal RemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = true

Scenario 1-7 (Add a New Non-Active RD on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("RD3-A", "4081003333", false) on TermA.	CiscoProvTerminal RemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false

Scenario 1-8 (Add a New Active RD on a 'CTI Remote Device', with Another Existing Active RD)

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("RD3-A", "4081003333", true) on TermA.	CiscoProvTermina lRemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = true

Scenario 1-9 (Add a New RD on a 'CTI Remote Device' with a Number That Is the Same as Another Existing RD's Number)

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("AnyName", "4081002222", false) on TermA.	Caught exception: com.cisco.jtapi. PlatformException Impl: Duplicated Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_DUPLICATED_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-10 (Remove a RD From a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes removeRemoteDestination("4081002222") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeRemoteDestination ("4081002222") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true

Scenario 1-11 (Remove All RD(s) From a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes removeAllRemoteDestinations() on terminal A.

Note JTAPI will loop through the terminal/device's existing remote destinations one by one, so the total number of CiscoProvTerminalRemoteDestinationChangedEv sent to an application should be the same number of available remote destinations being removed. And the order and content of each event can vary, depending on how each remote destination is stored in JTAPI's local cache RD list.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemote Destinations() on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false
	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = null.

Scenario 1-12 (Update a RD Name on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName ("4081001111", "MyHome") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4081001111", "MyHome") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-13 (Update a RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationNumber ("4081001111", "6268210080") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4081001111", "6268210080") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "6268210080" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-14 (Add a New RD with an Invalid RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination ("iPhone5", "IAmNotANumber", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("iPhone5", "IAmNotANumber", true) on TermA.	Caught exception: com.cisco.jtapi. PlatformExceptionImpl: Invalid Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-15 (Update RD Name with an Invalid/Not-Associated RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName ("4085268222", "MyBossOffice") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4085268222", "MyBossOffice") on TermA.	Caught exception: com.cisco.jtapi. PlatformExceptionImpl: Invalid Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-16 (Update RD Name with a Null RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName (null, "MyBossOffice") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name (null, "MyBossOffice") on TermA.	Caught exception: com.cisco.jtapi. InvalidArgument ExceptionImpl: Invalid Remote Destination Number/Name (updateRemoteDestination Name parameter).	TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-17 (Clear an Existing Active RD as a Non-Active RD on a 'CTI Remote Device')

Explicit Pre-condition: (RD1-A: "4081001111", True; RD2-A: "4081002222", False; RD3-A: "4081003333", False)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.setActiveRemoteDestination("4081001111", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("4081001111", false) on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false

Scenario 1-18 (Remove All RD(s) From a 'CTI Remote Device')

User1 Has "CTI Remote Device C" in the Control List. User Invokes removeAllRemoteDestinations() on Terminal C.

Action	Events	Call info5
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemoteDestinations() on TermC.		**Note** Nothing is removed as there is no RD on this device. JTAPI won't be sending any request to CTI. No CiscoJtapiException will be thrown either.

Scenario 1-19 (Remove All 5 RD(s) From a 'CTI Remote Device')

Explicit Pre-condition: RD1-A: "4081001111", true; RD2-A: "4081002222", false; RD3-A: "4081003333", false; RD4-A: "4081004444", false; RD5-A: "4081005555", false.

User1 has "CTI Remote Device A" in the control list. User invokes removeAllRemoteDestinations() on terminal A.

Note that JTAPI will loop through the terminal/device's existing remote destinations one by one, so the total number of CiscoProvTerminalRemoteDestinationChangedEv sent to an application should be the same number

of available remote destinations being removed. And the order and content of each event can vary, depending on how each remote destination is stored in JTAPI's local cache RD list.

Also note currently there is no checking in JTAPI to limit only up to 5 RDs per CTI Remote Device. If application tries to add a new RD to an existing CTI Remote Device that already has 5 RDs, JTAPI will simply send the add request to CTI and let it decide on pass/fail.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemote Destinations() on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[4]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false CiscoRemoteDestinationInfo[3].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[3].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[3].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = null.

Scenario 1-20 (Update a RD's Name and Number and Set It as ActiveRD on a 'CTI Remote Device' at the Same Time)

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestination ("4081002222", "MyVacationHome", "4081009999", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("4081002222", "MyVacationHome", "4081009999", true) on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "MyVacationHome" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "MyVacationHome" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081009999" CiscoRemoteDestinationInfo[1].getIsActiveRD() = true

CTI Remote Device Use Cases Group 2

Use Cases Group 2: CTIRD Incoming/Outgoing/Disconnect/Redirect/Hold/Resume and Shared-Line Call Scenarios

Pre-conditions on Use Cases group 2 below with default jtapi.ini settings, unless specified explicitly. Note that the CTI Ports have Auto-Accept enabled:

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD1", Line A (DN: 8881000))
- Remote Destination 1 (Name: "IRVOffice", Number: "919498231202", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort1", Line B (DN: 8881000))
- Device C (CTI Port - Name: "irvCTIPort6", Line C (DN: 8886000))
- Device D (CTI Port - Name: "irvCTIPort7", Line C (DN: 8887000))
- Device E (CTI Remote Device - Name: "irvCTIRD2", Line E (DN: 8889000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device F (CTI Remote Device - Name: "irvCTIRD3", Line E (DN: 8889001))
- Remote Destination 1 (Name: "IRVCell2", Number: "916267829526", Active RD: true)

Scenario 2-1 (Incoming Call From CTI Port to CTI Remote Device)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-2 (Incoming Call From CTI Port to Non-Observed CTI Remote Device)

C calls E, Application is observing C only on address and terminal. No observer on E. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-3 (Incoming Call From CTI Port to CTI Remote Device, but No Answer on Remote Destination)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 does not answers the call and time out.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEv 8889000 GC1: CallObservationEndedEv	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-4 (Incoming Call From CTI Port to CTI Remote Device, and Redirect to Another CTI Port)

C calls E, and E redirects the call to D, Application is observing all C, D, E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes connection on irvCTIRD2.redirect (8887000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8887000, null, REDIRECT_WITHOUT_ MODIFIED_CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
irvCTIPort7 answers the call.	GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000

Scenario 2-5 (Incoming Call From CTI Port to CTI Remote Device, and Redirect to Another CTI Remote Device)

C calls E, and E redirects the call to F, and C redirect the call to E. Application is observing all C, E, F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes connection on irvCTIRD2.redirect(8889001, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8889001, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
User1 invokes connection on irvCTIPort6.redirect(8889000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8889000, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889001 :: LastRedirectedPartyAddress: 8886000

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889001 :: LastRedirectedPartyAddress: 8886000

Scenario 2-6 (Incoming Call From CTI Port to CTI Remote Device with a Shared-Line of Another CTI Port)

C calls A (with a shared line with B), Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CallingAddress = 8886000, CalledAddress = 8881000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8881000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
Disconnect the call from irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-7 (Incoming Call From CTI Port to CTI Port with a Shared-Line of a CTI Remote Device)

C calls B (with a shared line of A), Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CallingAddress = 8886000, CalledAddress = 8881000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8881000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort1 answers the call.	GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnTalkingEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
Disconnect the call from irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-8 (Outgoing Call From CTI Remote Device to CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings and answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-9 (Outgoing Call From CTI Remote Device (with a Shared Line of CTI Port) to Another CTI Port)

A (with a shared line of B) calls D, Application is observing both A, B, and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD1, 8881000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000	CallingAddress = Unknown, CalledAddress = 8881000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8881000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1	
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1	CallingAddress = Unknown, CalledAddress = 8881000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8881000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CallingAddress = Unknown, CalledAddress = 8887000, CurrentCallingAddress = 8881000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8881000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
Disconnect the call from 8881000 connection.	GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress

Scenario 2-10 (Outgoing Call From CTI Remote Device to CTI Port, but No Answer on Active Remote Destination)

E calls D, Application is observing both E and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 does not answer the call and time out.	GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-11 (Outgoing Call From CTI Remote Device to CTI Port, but No Answer on CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress
irvCTIPort7 does not answer the call, time out.	GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: ConnFailedEvent 8889000 GC1: CallCtlConnFailedEv 8889000 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-12 (Outgoing Call From Non-Observed CTI Remote Device to CTI Port)

E calls D, Application is observing only on D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
From another provider, User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: Unknown :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnCreatedEvent 8889000 GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIPort7 answers the call.	GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-13 (Outgoing Call From CTI Remote Device to Non-Observed CTI Port)

E calls D, Application is observing only on E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
From another provider, User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings and answers the call from another provider.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-14 (Outgoing Call From CTI Remote Device to Another CTI Remote Device)

E calls F, Application is observing both E and F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8889001).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-15 (Outgoing Call From CTI Remote Device to Another CTI Remote Device, Then Redirect Again to a Third CTI Remote Device with a Shared-Line)

E calls F, then F redirect to A (with a shared-line with B), Application is observing all A, B, E and F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8889001).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Action	Events	Call info
User invokes connection on irvCTIRD3.redirect(8881000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8881000, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1). Both irvCTIRD1 and irvCTIPort1 are ringing.	GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001
irvCTIRD1's Active remote destination of 919498231202 answers the call. Terminal connection of irvCTIRD1 goes to 'talking' and irvCTIPort1 goes to 'bridged'.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Action	Events	Call info
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Scenario 2-16 (Disconnect an Incoming Call on CTI Remote Device After Answer While Talking)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User invokes connection.disconnect on irvCTIRD2 while talking.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-17 (Disconnect an Incoming Call on CTI Remote Device After Answer While Talking)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invokes connection.disconnect on irvCTIRD2 while it's still ringing on Active remote destination of 16267829523.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: ConnFailedEvent 8886000 GC1: CallCtlConnFailedEv 8886000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-18 (Disconnect an Outgoing Call From CTI Remote Device to CTI Port; Disconnect After Answering on Remote Destination and Answering on Called CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIPort7 rings and answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Action	Events	Call info
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-19 (disconnect an Outgoing Call From CTI Remote Device to CTI Port; Disconnect After Answering on Remote Destination but Before Answering on Called CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-20 (Disconnect an Outgoing Call From CTI Remote Device to CTI Port; Drop the Call Before Even Answering on Remote Destination. Note That Only One Connection on CTI Remote Device Which Is in Offering State)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 rings, User1 drops the call to disconnect from 8889000 connection.	GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-21 (Incoming Call From CTI Port to CTI Remote Device with a Shared-line of Another CTI Port). Note That irvCTIRD1 Remote Destination Answers the Call; Hold irvCTIRD1, Unhold irvCTIRD1; Then Hold irvCTIRD1, Unhold irvCTIPort1 Which Results irvCTIRD1 Got Disconnected; Then Hold irvCTIPort6, Unhold irvCTIPort6, Then Disconnect 8886000)

C calls A (with a shared line with B) with several hold/resume operations on different terminals. Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnHeldEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User invoke unhold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnHeldEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke unhold on terminalconnection of irvCTIPort1 (while it's in Bridged state). This results in irvCTIRD1 being dropped.	GC1: CallCtlTermConnTalkingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIPort6	GC1: CallCtlTermConnHeldEv irvCTIPort6	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke unhold on terminalconnection of irvCTIPort6	GC1: CallCtlTermConnTalkingEv irvCTIPort6	
Disconnect the call from connection of irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-22 (Outgoing Call From CTI Remote Device (with a Shared Line of CTI Port) to Another CTI Port). Note That irvCTIPort1 Rings and Remote Destination on irvCTIRD1 Rings, Remote Destination Answers the Call, Call Is Then Offered on irvCTIPort7. After Answer on irvCTIPort7, Disconnect From 8881000 Connection)

A (with a shared line of B) calls D, then with several hold/unhold operations at different terminals. Application is observing both A, B, and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD1, 8881000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000	CurrentCalledAddress: 8881000:: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1	
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1	CurrentCalledAddress: 8881000:: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: LastRedirectedPartyAddress: 8887000
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000
User invoke unhold on terminalconnection of irvCTIPort1 (while it's in Bridged state). This results in irvCTIRD1 being dropped.	GC1: CallCtlTermConnHeldEv irvCTIPort1 GC1: CallCtlTermConnTalkingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000

Action	Events	Call info
Disconnect the call from 8881000 connection.	GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-23 (Superprovider Acquires a CTIRD That Is Not on User Control List)

User1 open a provider which can observe any terminal (User1 with "Standard CTI Allow Control of All Devices" role), and then acquire a CTI Remote Device "CTIRD_UP" that is not on User1's control list).

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes provider.createTerminal(CTIRD_UP).	CiscoAddrCreatedEv: 8889999 CiscoTermCreatedEv: CTIRD_UP	

CTI Remote Device Use Cases Group 3

Use Cases Group 3 (CUCSF Registration and Unregistration, for Normal SIP Mode <-> Extend Mode, and Terminal Switching Scenarios)

Pre-conditions on Use Cases group 3 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state.
- Device A (CUCSF - Name: "irvCSF1", Line A (DN: 7771000))
- Remote Destination 1 (Name: "IRVCell", Number: "916267829523", Active RD: true)
- Scenario 3-1 (Registration of CUCSF in between Extend mode and SIP mode).:

Device A is registered in normal SIP mode when Webex with Jabber client is running and configured this device as its associated phone. User1 open provider and add observers on this device to bring it in service. Now exit Webex to unregister it from SIP, and then User1 calls CiscoTerminal.register() to register it to Extend mode from JTAPI, then add observers back to bring it in service. Now User1 unregister it from Extend mode in JTAPI by calling CiscoRemoteTerminal.unregister(). Now open Webex again to register this device back to SIP mode, and add observers back to bring it in service.

Action	Events	Info
Webex is opened. User1 adds provider observer. registerFeature 1235 on provider. addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: ProvInServiceEv 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = true CiscoRemoteTerminal.getType() = 503 CiscoRemoteTerminal.getTypeName() = Cisco Unified Client Services Framework
Exit Webex	Provider: CiscoProvTerminalUnRegisteredEv irvCSF1 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermOutOfServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = false
User1 calls CiscoTerminal.register() on irvCSF1. addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: CiscoAddrRemovedEv 7771000 Provider: CiscoTermRemovedEv irvCSF1 Provider: CiscoAddrCreatedEv 7771000 Provider: CiscoTermCreatedEv irvCSF1 Provider: CiscoProvTerminalRegisteredEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrOutOfServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 3 CiscoTerminal.isRegistered() = true CiscoRemoteTerminal.getRegistrationType() = 8 CiscoRemoteTerminal.isRegisteredByThisApp() = true
User1 calls CiscoRemoteTerminal. unregister()	7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermOutOfServiceEv Provider: CiscoProvTerminalUnRegisteredEv	CiscoTerminal.getProtocol = 3 CiscoTerminal. isRegistered() = false CiscoRemoteTerminal.getRegistrationType() = -1 CiscoRemoteTerminal. isRegisteredByThisApp() = false

Action	Events	Info
Open Webex addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: CiscoAddrRemovedEv 7771000 Provider: CiscoTermRemovedEv irvCSF1 Provider: CiscoAddrCreatedEv 7771000 Provider: CiscoTermCreatedEv irvCSF1 Provider: CiscoProvTerminalRegisteredEv 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = true

CTI Remote Device Use Cases Group 4

Use Cases Group 4 (Set/Reset Active Remote Destination Scenarios)

Pre-conditions on Use Cases group 4 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state. Single node.
- Device A (CTI Remote Device - Name: "irvCTIRD2", Line A (DN: 8881000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: false)
- Scenario 4-1 (User1 opens provider P1, set RD1 as active. Now User1 opens another provider P2, and set same RD1 as active again. Now stop CTI Manager service in this single node, the active RD would be clear out. Now restart CTI Manager, JTAPI will do a provider retry, and upon successfully connection, it will automatically reset the same RD1 as active again seamlessly.):

Action	Events	Info
User1 open Provider P1 and adds provider observer.	P1: ProvInServiceEv	
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P1.	P1: CiscoProvTerminalRemoteDestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true
User1 open Provider P2 and adds provider observer.	P2: ProvInServiceEv	

Action	Events	Info
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P2.	P1: CiscoProvTerminalRemote DestinationChangedEv P2: CiscoProvTerminalRemote DestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : false CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = false P2 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true
Stop CTI Manager on this single node where P1 & P2 are connected to. And this active RD will be clear out automatically from CTI/CCM side.	P1: ProvOutOfServiceEv P2: ProvOutOfServiceEv	
Start this CTI Manager. And JTAPI will automatically reset the same RD1 as active again seamlessly.	P1: ProvInServiceEv P2: ProvInServiceEv	Note that no CiscoProvTerminalRemote DestinationChangedEv will be sent to application because it is the same active RD that application previously set.

Scenario 4-2 (User1 Opens Provider P1, Add All Observers on Provider, Terminals, Addresses, Then Set RD1 as Active. Now Stop CTI Manager Service, the Active RD Would Be Clear Out. Now Restart CTI Manager, JTAPI Will Do a Provider Retry, and Upon Successfully Connection, It Will Automatically Reset the Same RD1 as Active Again Seamlessly)

Action	Events	Info
User1 open Provider P1 and adds provider observer.	P1: ProvInServiceEv	
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P1.	P1: CiscoProvTerminalRemote DestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true

Action	Events	Info
Stop CTI Manager where P1 is connected to. And this active RD will be clear out automatically from CTI/CCM side.	8881000:: Event: CiscoAddrOutOfServiceEv irvCTIRD2:: Event: CiscoTermOutOfServiceEv P1: ProvOutOfServiceEv	
Start this CTI Manager. And JTAPI will automatically reset the same RD1 as active again seamlessly.	P1: ProvInServiceEv irvCTIRD2:: Event: CiscoTermInServiceEv 8881000:: Event: CiscoAddrInServiceEv	Note that no CiscoProvTerminalRemote DestinationChangedEv will be sent to application because it is the same active RD that application previously set.

CTI Remote Device Use Cases Group 5

Use Cases Group 5 (CTIRD Transfer/Conference/Multiple-Calls Call Scenarios)

Pre-conditions on Use Cases group 5 below with default jtapi.ini settings, unless specified explicitly (Note: The CTI Ports have Auto-Accept enabled):

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD2", Line A (DN: 8889000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort4", Line B (DN: 8884000))
- Device C (CTI Port - Name: "irvCTIPort5", Line B (DN: 8884000))
- Device D (CTI Port - Name: "irvCTIPort6", Line D (DN: 8886000))
- Device E (CTI Remote Device - Name: "irvCTIRD3", Line E (DN: 8889001))
- Remote Destination 1 (Name: "IRVHome1", Number: "916268210080", Active RD: true)

Scenario 5-1 (Direct Transfer on CTI Remote Device to CTI Port)

D calls A with GC1 as GCID of call; A calls B with GC2 as GCID of call. Set A as transfer controller, and then transfer call from GC2 to GC1. Application is observing all A, B, D.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIRD2, 8889000, 8884000) , and answer() on irvCTIPort4 terminal connection.	GC1: CallCtlTermConnHeldEv irvCTIRD2 GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnInitiatedEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2 GC2: CallCtlConnDialingEv : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: ConnCreatedEvent : Address: 8884000 GC2: ConnInprogressEvent : Address: 8884000 GC2: CallCtlConnOfferedEv : Address: 8884000 GC2: ConnAlertingEvent : Address: 8884000 GC2: CallCtlConnAlertingEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort4 GC2: TermConnRingingEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIPort4 GC2: TermConnCreatedEvent : Terminal: irvCTIPort5 GC2: TermConnRingingEvent : Terminal: irvCTIPort5	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8884000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8884000, No LastRedirectedPartyAddress

Action	Events	Call info
	GC2: CallCtlTermConnRingingEv : Terminal: irvCTIPort5 GC2: ConnConnectedEvent : Address: 8884000 GC2: CallCtlConnEstablishedEv : Address: 8884000 GC2: TermConnActiveEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort4 GC2: TermConnPassiveEvent : Terminal: irvCTIPort5 GC2: CallCtlTermConnInUseEv : Terminal: irvCTIPort5	
User1 invokes GC2.setTransferController (terminal connection of irvCTIRD2).	GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2 GC1: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000 CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes GC2.transfer(GC1).	GC2: CiscoTransferStartEv GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC1: CiscoCallChangedEv GC2: ConnCreatedEvent : Address: 8886000 GC2: ConnConnectedEvent : Address: 8886000 GC2: CallCtlConnEstablishedEv : Address: 8886000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort6 GC2: TermConnActiveEvent : Terminal: irvCTIPort6 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort6 GC1: TermConnDroppedEv : Terminal: irvCTIPort6	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000
	GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Action	Events	Call info
disconnect() the call on 8886000 connection.	GC2: TermConnDroppedEv irvCTIPort6 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC2: ConnDisconnectedEvent : Address: 8886000 GC2: CallCtlConnDisconnectedEv : Address: 8886000 GC2: TermConnDroppedEv : Terminal: irvCTIPort4 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort4 GC2: TermConnDroppedEv : Terminal: irvCTIPort5 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort5 GC2: ConnDisconnectedEvent : Address: 8884000 GC2: CallCtlConnDisconnectedEv : Address: 8884000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000

Scenario 5-2 (Conference Call on CTI Remote Device and CTI Port with Another CTI Remote Device)

D calls A with GC1 as GCID of call; A calls E with GC2 as GCID of call. Set A as conference controller, and then conference/join call from GC2 to GC1. Application is observing all A, D, E.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIRD2, 8889000, 8889001).	GC1: CallCtlTermConnHeldEv irvCTIRD2 GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnInitiatedEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2 GC2: CallCtlConnDialingEv : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: ConnCreatedEvent : Address: 8889001 GC2: ConnInprogressEvent : Address: 8889001 GC2: CallCtlConnOfferedEv : Address: 8889001 GC2: ConnAlertingEvent : Address: 8889001 GC2: CallCtlConnAlertingEv : Address: 8889001 GC2: TermConnCreatedEvent : Terminal: irvCTIRD3 GC2: TermConnRingingEvent : Terminal: irvCTIRD3 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIRD3	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
irvCTIRD3's Active remote destination of 916268210080 answers the call.	GC2: ConnConnectedEvent : Address: 8889001 GC2: CallCtlConnEstablishedEv : Address: 8889001 GC2: TermConnActiveEvent : Terminal: irvCTIRD3 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD3	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000
User1 invokes GC2.setConference Controller (terminal connection of irvCTIRD2).	GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2 GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2	

Action	Events	Call info
User1 invokes GC2.conference(GC1).	GC2: CiscoConferenceStartEv GC1: TermConnDroppedEv : Terminal: irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC1: CiscoCallChangedEv GC2: ConnCreatedEvent : Address: 8886000 GC2: ConnConnectedEvent : Address: 8886000 GC2: CallCtlConnEstablishedEv : Address: 8886000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort6 GC2: TermConnActiveEvent : Terminal: irvCTIPort6 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort6 GC1: TermConnDroppedEv : Terminal: irvCTIPort6 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000

Action	Events	Call info
disconnect() the call on 8889000 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIRD3 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD3 GC2: ConnDisconnectedEvent : Address: 8889001 GC2: CallCtlConnDisconnectedEv : Address: 8889001	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000
disconnect() the call on 8889001 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIPort6 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC2: ConnDisconnectedEvent : Address: 8886000 GC2: CallCtlConnDisconnectedEv : Address: 8886000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889001, LastRedirectedPartyAddress: 8889000

Scenario 5-3 (Multiple Calls on CTI Remote Device)

D calls A with GC1 as GCID of call; B calls A with GC2 as GCID of call. Application is observing all A, B, D.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIPort4, 8884000, 8889000)	GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8884000 GC2: ConnConnectedEvent : Address: 8884000 GC2: CallCtlConnInitiatedEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort4 GC2: TermConnActiveEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort4 GC2: CallCtlConnDialingEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort5 GC2: TermConnPassiveEvent : Terminal: irvCTIPort5 GC2: CallCtlTermConnInUseEv : Terminal: irvCTIPort5 GC2: CallCtlConnEstablishedEv : Address: 8884000 GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnInprogressEvent : Address: 8889000 GC2: CallCtlConnOfferedEv : Address: 8889000 GC2: ConnAlertingEvent : Address: 8889000 GC2: CallCtlConnAlertingEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnRingingEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIRD2	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
User1 invokes GC2.answer(terminal connection of irvCTIRD2).	GC1: CallCtlTermConnHeldEv : Terminal: irvCTIRD2 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
disconnect() the GC2 call on 8889000 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIPort4 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort4 GC2: TermConnDroppedEv : Terminal: irvCTIPort5 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort5 GC2: ConnDisconnectedEvent : Address: 8884000 GC2: CallCtlConnDisconnectedEv : Address: 8884000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
User1 invokes unhold() on GC1 terminal connection of irvCTIRD2. disconnect() the GC1 call on 8889000 connection.	GC1: CallCtlTermConnTalkingEv GC1: TermConnDroppedEv : Terminal: irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC1: TermConnDroppedEv : Terminal: irvCTIPort6 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

CTI Remote Device Use Cases Group 6

Use Cases Group 6 (CTIRD URI-Dialing Basic Incoming and Outgoing DVO Call Scenarios)

Pre-conditions on Use Cases group 6 below with default jtapi.ini settings, unless specified explicitly (Note: The CTI Ports have Auto-Accept enabled):

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD3", Line A (DN: 8889001, Directory URIs: "8889001A@cisco.com"))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort2", Line B (DN: 8882000, Directory URIs: "8882000A@cisco.com"))

Scenario 6-1 (Basic Incoming Call From CTI Port to CTI Remote Device Via URI)

B calls A with GC1 as GCID of call. Application is observing both A and B.

Action	Events	Call info
User1 invokes call.connect (irvCTIPort2, 8882000, "8889001A@cisco.com")	GC1: CallActiveEvent GC1: ConnCreatedEvent 8882000 GC1: ConnConnectedEvent 8882000 GC1: CallCtlConnInitiatedEv 8882000 GC1: TermConnCreatedEvent irvCTIPort2 GC1: TermConnActiveEvent irvCTIPort2 GC1: CallCtlTermConnTalkingEv irvCTIPort2 GC1: CallCtlConnDialingEv 8882000 GC1: CallCtlConnEstablishedEv 8882000 GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
Disconnect() the call on 8882000 connection.	GC1: TermConnDroppedEv irvCTIPort2 GC1: CallCtlTermConnDroppedEv irvCTIPort2 GC1: ConnDisconnectedEvent 8882000 GC1: CallCtlConnDisconnectedEv 8882000 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Scenario 6-2 (Basic Outgoing DVO Call From CTI Remote Device to CTI Port Via URI)

A calls B with GC1 as GCID of call. Application is observing both A and B.

Action	Events	Call info
User1 invokes call. connect (irvCTIRD3, 8889001, "8882000A@cisco.com")	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889001, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3 GC1: ConnCreatedEvent 8882000 GC1: ConnInprogressEvent 8882000 GC1: CallCtlConnOfferedEv 8882000 GC1: ConnAlertingEvent 8882000 GC1: CallCtlConnAlertingEv 8882000 GC1: TermConnCreatedEvent irvCTIPort2 GC1: TermConnRingingEvent irvCTIPort2 GC1: CallCtlTermConnRingingEv irvCTIPort2	CallingAddress = Unknown, CalledAddress 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress
Answer() the call on 8882000 terminal connection.	GC1: ConnConnectedEvent 8882000 GC1: CallCtlConnEstablishedEv 8882000 GC1: TermConnActiveEvent irvCTIPort2 GC1: CallCtlTermConnTalkingEv irvCTIPort2	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress
Disconnect() the call on 8889001 connection.	GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnDroppedEv irvCTIPort2 GC1: CallCtlTermConnDroppedEv irvCTIPort2 GC1: ConnDisconnectedEvent 8882000 GC1: CallCtlConnDisconnectedEv 8882000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress

CTI RD Call Forward

Table 264: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2, E - Enterprise Line

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observer on A, B and E	ProvInServiceEv	
A calls B	CallActiveEv on A, B, E ConnCreatedEv on A, B, E ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, E TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnCreatedEv on B ConnInProgressEv on B, E CallCtlConnOfferedEv on B, E ConnAlertingEv on A, B, E CallCtlConnAlertingEv on A, B, E TermConnCreated on Term B TermConnRingingEv on B, E CallCtlTermConnRingingEv on B, E	All RDD's will ring
C answers the call	**At Step 3:** ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	

Action	Events	Call Info
Disconnects the call	At Step 4: TermConnDroppedEv on A, B, E CallCtlTermConnDroppedEv on A, B, E ConnDisconnectedEv on A, B, E CallCtlConnDisconnectedEv on A, B, E CallInvalidEv CallObservationEndedEv on A, B, E	

Table 265: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected "is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Opens only A and no events for B	ProvInServiceEv	
GC1: A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B, ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B, ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will disconnect with message USER_BUSY	ConnFailedEv for A	USER_BUSY on Shared enterprise line

Table 266: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A- IP Phone, B - CTI-RD. VoiceMail Is Configured

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
GC1: A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnInProgressEv on VoiceMail of B CallCtlConnOfferedEv on VoiceMail of B ConnAlertingEv on VoiceMail of B CallCtlConnAlertingEv on VoiceMail of B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 267: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A - IP Phone, B - CTI-RD. VoiceMail Is Configured for B

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Only A is observed	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, C CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 268: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A IP Phone, B CTI-RD, C RDD1, D RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Applications adds C as the active remote destination on B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on CallCtlTermConnRingingEv on B	
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B CallCtlTermConnDroppedEv on A, B ConnDisconnectedEv on A, B CallCtlConnDisconnectedEv on A, B CallObservationEndedEv on A, B	

Table 269: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A IP Phone, B CTI-RD, C RDD1, D RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Applications adds C as the active remote destination on B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B CallCtlTermConnDroppedEv on A, B ConnDisconnectedEv on A, B CallCtlConnDisconnectedEv on A, B CallObservationEndedEv on A, B	

Table 270: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2, E - Enterprise Line

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observer on A, B and E	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, E ConnCreatedEv on A, B, E ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, E TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, E CallCtlConnAlertingEv on A, B, E ConnInProgressEv on B, E CallCtlConnOfferedEv on B, E TermConnRingingEv on B, E CallCtlTermConnRingingEv on B, E	All RDD will ring
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B, E CallCtlTermConnDroppedEv on A, B, E ConnDisconnectedEv on A, B, E CallCtlConnDisconnectedEv on A, B, E CallObservationEndedEv on A, B, E	

Table 271: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected "is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B, ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B, ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	All RDD will ring
call will disconnect with message USER_BUSY		USER_BUSY on Shared enterprise line

Table 272: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 273: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, C CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

CTI Video Support

Use cases related to CTI Video Support feature are mentioned below:

Scenario 1:

Phone A is video capable, telepresence capable, with 1 screen and a camera, and in registered state. User1 has phone A in the control list. User invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() before opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
User1 invokes CiscoTerminal.i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() = TELEPRESENCEINTEROP_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 2

Phone A is not video capable, not telepresence capable with 0 screens. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() before opening device

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 3

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() after opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 4

Phone A is video not capable, not telepresence capable with 0 screens. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() after opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 5

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 does not have phone A in the control list. User1 has Super provider capabilities.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 aquires phone A using prov.createTerminal("phoneA")	CiscoTermCreatedEv TermA CiscoAddrCreatedEv A	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_ENABLED
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 6

Phone A is not video capable, not telepresence capable and has 0 screens. User1 does not have phone A in the control list. User1 has Super provider capabilities

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 aquires phone A using prov.createTerminal("phoneA")	CiscoTermCreatedEv TermA CiscoAddrCreatedEv A	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 7

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registeres it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario 8

Basic Video call: Phone A is video enabled, telepresence enabled with 1 screen. Phone B is video disabled , telepresence disabled with 0 screens. Both the phones are in the control list of User1.

Action	Events	Call info
User Opens provider and adds observer	ProvInServiceEv	
User adds terminal observers on Phone A and Phone B	CiscoTermInServiceEv TermA CiscotermInServiceEv TermB	
User adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call info
User makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1		The API returns 1, indicating video capable device(VIDEO_ENABLED) for TermA(far-end party).
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1		The API returns 1, indicating telepresence capable device(TELEPRESENCEINTEROP_ENABLED) for TermA(far-end party).
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo. getScreenCount() on GC1		The API returns 1, indicating device has 1 screen, for TermA(far-end party).
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1		The API returns 0, indicating video capable device(NONE) for Term B
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1		The API returns 1, indicating device is not telepresence capable (TELEPRESENCEINTEROP_NONE) for TermA(far-end party).
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo .getScreenCount() on GC1		The API returns 1, indicating device has 0 screens, for TermA(far-end party).

Scenario 9

Phone A is video disabled (in CUCM Admin Phone page, the Video Capabilities field is 'Disabled') , but the device has an an external camera (USB or CUVA) plugged in. Phone A is in registered state.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE

Action	Events	Call info
In Device Configuration CUCM Admin pages- Video Capabilities field is changed to'Enabled'	CiscoProvTerminalMultiMedia CapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
The external camera is removed, or the Cisco Camera field in CUCM Admin Phone page is 'Disabled'	No event is delivered, as the device is still a video capable device as it can receive video	
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
In Device Configuration CUCM Admin pages- Video Capabilities field is changed to'Disabled'	CiscoProvTerminalMultiMedia CapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE

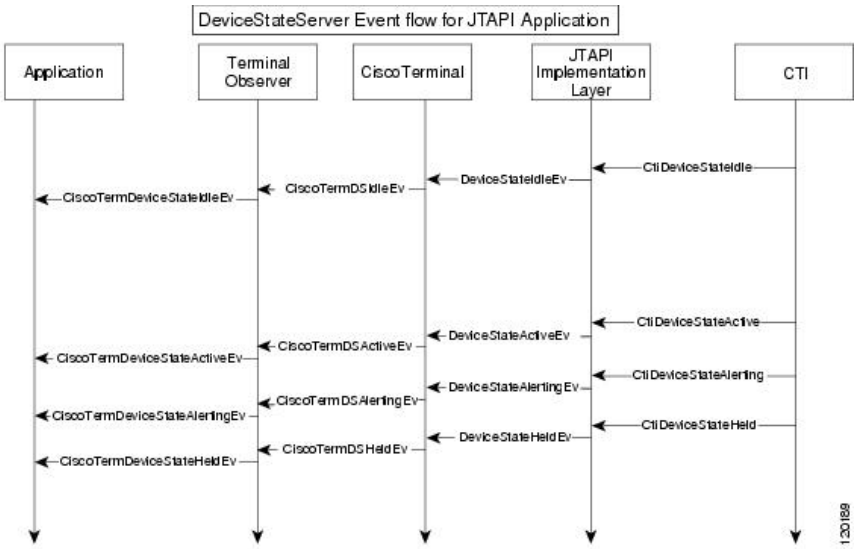
Device and Line Restriction

S.No	Scenario	Events
1	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list. T1 and A3 is added into the restricted list. Application opens the provider	
	Application queries for is Restricted on T1, T2, T3	CiscoTerminal.isRestricted() returns true for T1 and false for T2 and T3
	Application queries for is Restricted on Address A1, A2, A3	CiscoAddress.isRestricted() returns true for A1, A3, false for A2.
		CiscoAddress.getRestrictedAddrTerminals() on A1, A3 returns T1, T3 respectively, returns null for A2.
	Application tries to addObserver and addCallObserver on T1, T2, T3, A1, A2, A3	addObserver and addCallObserver fails for T1, A1, A3. For T3 observer is added, but no events are received on A3. For A2, application will be able to add observers successfully and events will be received
2	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.	

S.No	Scenario	Events
	Application opens the provider and adds observer on all terminals and addresses.	CiscoTermRestrictedEv for T1 CiscoAddrRestrictedEv for L1 CiscoAddrRestrictedEv for A2 sent to providerObserver.
	T1 and A2 are added to the restrictedlist.	CiscoTermOutOfServiceEv for T1 CiscoAddrOutOfServiceEv for L1 CiscoAddrOutOfServiceEv for A2
	T1 and L2 are removed from restricted list	CiscoTermActivatedEv for T1 and CiscoAddrActivatedEv for A1 CiscoAddrActivatedEv for A2 sent to providerObserver. CiscoTermInServiceEv for T1 and CiscoAddrInServiceEv for A1 CiscoAddrInServiceEv for A2 sent to terminal and address observers.
3	Application has Devices T1, T2, T3 whose lines are A1, A1, A2 in the control list. A1 is the shared line on T1 and T2 Application opens provider and adds observer on all terminals/addresses	
	T1 is added into the restricted list.	Application will see CiscoTermRestrictedEv for T1 and CiscoAddrRestrictedOnTerminalEv which contains getAddress is L1 and getTerminal as T1. Application will also see CiscoTermOutOfServiceEv for T1 and CiscoAddrOutOfService for A1/T1
	T1 is removed from the restricted list	CiscoTermActivatedEv for T1 CiscoAddrActivatedEv for L1 CiscoTermInServiceEv for T1 CiscoAddrInServiceEv for A1/T1
4	Application has Devices T1, T2, T3 whose lines are A1, A1, A1 in the control list. A1 is the shared line on T1, T2 and T3 Application opens the provider and adds observer on all terminals and addresses	
	A1 on T1 is added to the restricted list	CiscoAddrRestrictedOnTerminalEv for A1/T1 CiscoAddrOutOfServiceEv for A1/T1
	A1 on T2 is added to therestricted list	CiscoAddrRestrictedOnTerminalEv for A1/T2 CiscoAddrOutOfServiceEv for A1/T2
	A1 on T3 is added to therestricted list	CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1/T3

S.No	Scenario	Events
	A1 on T1 is removed from the restricted list	CiscoAddrActivatedOnTerminalEv for A1/T1 CiscoAddrInServiceEv for A1/T1
	A1 on T2 is removed from the restricted list	CiscoAddrActivatedOnTerminalEv for A1/T2 CiscoAddrInServiceEv for A1/T2
	A1 on T3 is removed from the restricted list	CiscoAddrActivatedEv for A1 CiscoAddrInServiceEv for A1/T3
5	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.	
	Application opens the provider and adds observer on all terminals and addresses. A1 is involved in a call with party X.	CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1
	A1 is added into the restricted list.	ConnDisconnectedEv CallCtlConnDisconnectedEv TermConnDroppedEv CallCtlConnDroppedEv CallInvalidEv

Device State Server



Do Not Disturb

Configuration: Application is observing terminal A and terminal B.

Scenario One

Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is enabled via setDNDChangedEvFilter. DND is enabled on the terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application adds terminal observer to terminal A. Filter is enabled via setDNDChangedEvFilter() in CiscoTermEvFilter. DND is enabled on the terminal through phone or admin page. Application invokes getDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to the application	N.A

Scenario Two

Application enables filter to receive events. Application adds Terminal observer to terminal A using Terminal.addObserver(). DND is enabled on the terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application enables filter to receive events. Application adds terminal observer to terminal A. DND is enabled on the device through phone or admin pages. Application invokes getDNDStatus() from CiscoTerminal.	NEW META EVENT_____META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to the application	N.A

Scenario Three

Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is disabled via setDNDChangedEvFilter() in CiscoTermEvFilter. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is disabled via setDNDChangedEvFilter() in CiscoTermEvFilter. Application invokes getDNDStatus() from CiscoTerminal.	CiscoTermDNDStatusChangedEv is not delivered to application.	N.A

Scenario Four

Application does not add Terminal observer to terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application does not add Terminal observer to terminal. Application invokes getDNDStatus() from CiscoTerminal.	InvalidStateException is thrown	N.A

Scenario Five

Application does not enable the filter to receive events. Application adds Terminal observer to terminal A. DND status is set to true through the phone or admin pages. Application now enables the filter to receive events. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application does not enable the filter to receive events.Application adds Terminal observer to terminal A. DND status is set to true through the phone or admin pages. Application now enables the filter to receive events Application invokes getDNDStatus() from CiscoTerminal.	CiscoTermDNDStatusChangedEv is not delivered to application. NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to application	N.A

Scenario Six

Application sets DND status to false by invoking the setDNDStatus() interface on CiscoTerminal.

Action	Events	Call info
Application invokes setDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A

Scenario Seven

Application 1 and Application 2 are observing terminal a, and both the applications have enabled the filter to receive events. Application 1 sets DND status to false on Terminal A. Application 2 is observing Terminal A.

Action	Events	Call info
Application invokes setDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A

Scenario Eight

DND Type is RingerOff and CFNA is not set. Terminal B calls Terminal A. Call is presented to A and call is not answered.

Action	Events	Call info
Application invokes redirect() API with feature priority set to 3 from CiscoCall.	Call is presented to the device, irrespective of the DND settings on the device. CER call overrides DND setting.	N.A
Application invokes selectRoute() API with feature priority set to 3 from CiscoRouteSession.	Call is presented to the device, irrespective of the DND settings on the device. CER call overrides DND setting.	N.A

Scenario Nine

Action	Events	Call info
DND Type is RingerOff and CFNA is not set. Terminal B calls Terminal A .Call is presented to A and call is not answered.	ConnFailedEv Cause: CAUSE_NO ANSWER	N.A

Scenario Ten

DND Type is CallReject and CFB is not set. Terminal B calls Terminal A. Call is not presented to A.

Action	Events	Call info
DND Type is CallReject and CFB is not set. Terminal B calls Terminal A. Call is not presented to A	ConnFailedEv Cause: CAUSE_USER BUSY	N.A

Scenario Eleven

DND is enabled on the terminal A. Terminal A comes IN_SERVICE. Application invokes getDNDStatus() on CiscoTerm in ServiceEv.

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE.	CiscoTermInServiceEv Cause: CAUSE_NORMAL DND Status = true	N.A

Scenario Twelve

DND is enabled on terminal A. Terminal A comes IN_SERVICE. Application invokes setDNDStatus(). DB failure happens after the setDNDStatus() request is sent.

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE. Application invokes setDNDStatus(). DB failure follows and value is not updated in DB.	PlatformException is thrown “Could not meet post conditions of setDNDStatus()” No CiscoTermDNDStatusChangedEv is received.	N.A

Scenario Thirteen

DND is enabled on the terminalA. Terminal A comes IN_SERVICE, DND status is currently true in phone/admin. Application tries to set the same value i.e. invokes setDNDStatus(true).

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE.DND status is currently true in phone/admin.Application tries to set the same value i.e. invokes setDNDStatus(true).	InvalidStateException is caught: DND status with value true is already set No CiscoTermDNDStatusChangedEv is received.	N.A

DND-R

Scenario One

Application adds Terminal observer to terminal A using Terminal.addObserver (). DND-R is enabled on the terminal B via the Admin page or the Common profile page.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page. A issues Call.connect to B with the feature Priority = 1 (Normal)	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL ConnFailedEv B Cause:. Cause: CAUSE_USERBUSY.	N.A

Scenario Two

Application adds Terminal observer to terminal A using Terminal.addObserver (). DND-R is enabled on the Terminal B via the Admin page or the Common profile page.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page. A issues Call.connect to B with the feature Priority = 3 (Emergency)	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause:CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause:CAUSE_NORMAL ConnInProgressEv for B Cause:CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause:CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A Called: B

Scenario Three

DND-Call reject with CFB not set.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page with no CFB Setting. Terminal A issues Call.connect to Terminal B.	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause:CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause:CAUSE_NORMAL ConnFailedEv B Cause:CAUSE_USERBUSY.	NA

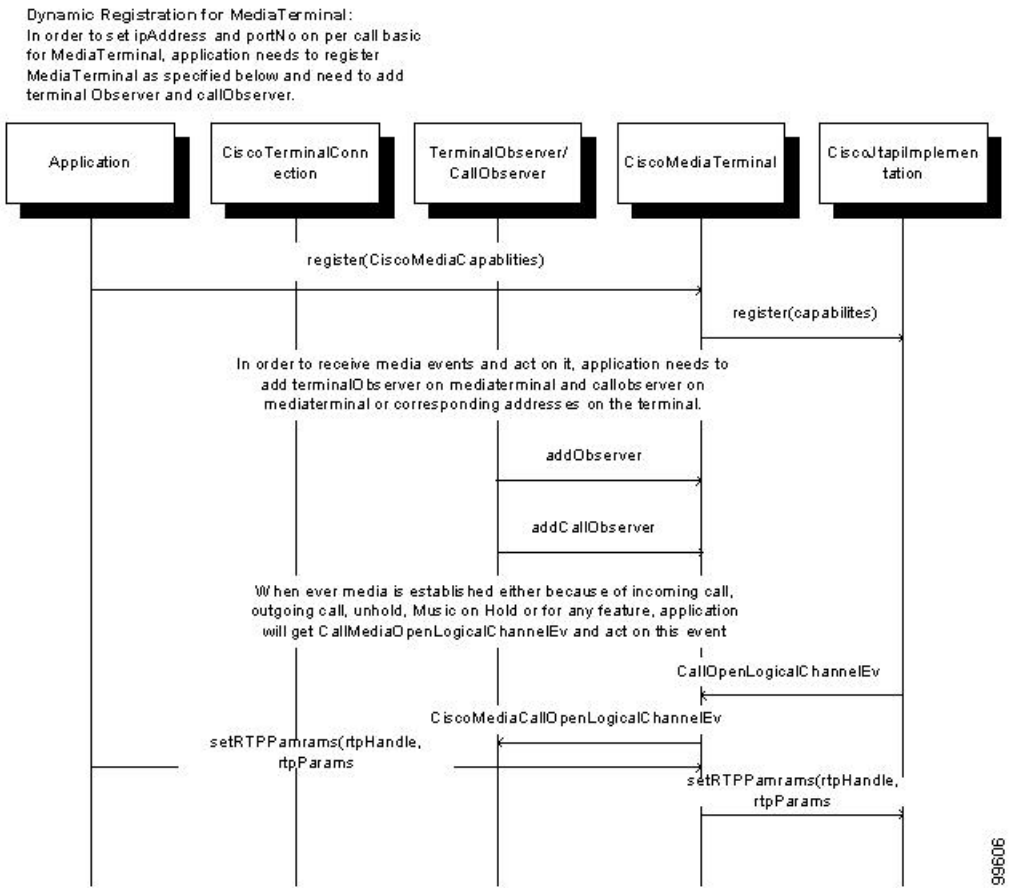
Scenario Four

DND – Call reject with CFB set to C.

Action	Events	Call info
Application is Observing Terminal A, B & C. DND-R is Enabled in Terminal B with CFB set to Terminal C. Terminal A issues Call.connect to Terminal B. Call is not Presented on Terminal B and is Forwarded to Terminal C.	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for C cause: REDIRECTED CALL ConnInProgressEv for C Cause: REDIRECTED CALL CallCtlConnOfferedEv for C Cause: REDIRECTED CALL ConnAlertingEv for C Cause REDIRECTED CALL CallCtlConnAlertingEv for C Cause: REDIRECTED CALL TermConnCreatedEv for C Cause: REDIRECTED CALL TermConnRingingEv for C Cause: REDIRECTED CALL CallCtlTermConnTalkingEv Cause: CAUSE_REDIRECTED	Calling: A Called: C LastRedirectedParty: B

Dynamic CTIPort Registration Per Call

The following diagram illustrates the message flows for Dynamic CTIPort Registration per call.



E911 Teleworker

SelectRoute Method

Use Case	Events	Call info
No changes in use case with selectRoute Method	No changes in the events	No changes in the callinfo

Redirect Method

Use Case	Events	Call info
No changes in use case with redirect Method	No changes in the events	No changes in the callinfo

Encryption Enhancement

Table 274: Service Parameter "Require Public Key Encryption" Is Set to "False". Application Is Using a Pre 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 275: Service Parameter "Require Public Key Encryption" Is Set to "True". Application Is Using a Pre 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider	PlatformException	getErrorCode() = CiscoJtapiException. INCOMPATIBLE_PROTOCOL_ VERSION

Table 276: Service Parameter "Require Public Key Encryption" Is Set to "False". Application Is Using a 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 277: Service Parameter "Require Public Key Encryption" Is Set to "True". Application Is Using a 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 278: SP Is Set to "True". Application Is Using 10.x CiscoJTAPI Lib. Application Has Provided Pub and Sub Ctimanager IP in the ProviderString

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	
CTIManager on the server to which app is connected goes down	ProvOutOfService	

End to End Call Tracing

Actions	Events	Call info
1.a) Both A and B are in user's control list. Basic Call A calls B; App is observing only B	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A … … GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlTermConnTalkingEv for TB	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID1 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(TB) returns ID1 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(term) will throw InvalidStateException
1.b) Only B is in User's control list Basic Call A calls B; App is observing only B	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A …… …… GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlTermConnTalkingEv for TB	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID1 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(TB) returns ID1 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(term) will throw PrivilegeViolationException

Actions	Events	Call info
2.a) Redirect in offering state Scenario All Observed A calls B; B redirects the call to C in offering state C Accepts the call C answers the call	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlConnEstablishedEv for C GC1: CallCtlTermConnTalkingEv for TC	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID2 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID3 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID4

Actions	Events	Call info
2.b) Redirect in connected state Scenario All Observed A calls B B answers B redirects the call to C in connected state	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID6 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID7 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID8

Actions	Events	Call info
3. Redirect Scenario Only B & C are Observed A calls B; B answers B redirects the call to C in connected state	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlConnEstablishedEv for A GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnTalkingEv for TB GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID10 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) throws InvalidStateException ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID11

Actions	Events	Call info
4. Conference Scenario; All observed GC1: A calls B; B answers GC2: B calls C; C answers GC1.conference(GC2)		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID12 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID13 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID14 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID15 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID16

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnHeldEv for TB GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for TB GC2: TermConnActiveEvent for TB GC2: CallCtlTermConnTalkingEv for TB GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnOfferedEv for C GC2: TermConnCreatedEv for TC GC2: ConnConnectedEv for B GC2: CallCtlConnEstablishedEv for B GC2: ConnConnectedEv for C GC1: CiscoConferenceStartEv GC1: CiscoCallFeatureCancelledEv GC2: CiscoCallChangedEv	

Actions	Events	Call info
	GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoConferenceEndEv	

Actions	Events	Call info
5. Transfer Scenario; All observed GC1: A calls B; B answers GC2: B calls C; C answers GC1.transfer(GC2)		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID19 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID20 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID21 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID22 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID23

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnHeldEv for TB GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for TB GC2: TermConnActiveEvent for TB GC2: CallCtlTermConnTalkingEv for TB GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnOfferedEv for C GC2: TermConnCreatedEv for TC GC2: ConnConnectedEv for B GC2: CallCtlConnEstablishedEv for B GC2: ConnConnectedEv for C GC2: CallCtlConnEstablishedEv for C GC1: CiscoTransferStartEv GC2: CiscoCallChangedEv	

Actions	Events	Call info
	GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	

Actions	Events	Call info
6. Shared Line Scenario; All Observed; DN B is present on T1, T2 A calls B; B(T1) Answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: TermConnCreatedEv for T1B GC1: TermConnRingingEv for T1B GC1: CallCtlTermConnRingingEv for T1B GC1: TermConnCreatedEv for T2B GC1: TermConnRingingEv for T2B GC1: CallCtlTermConnRingingEv for T2B GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for T1B GC1: CallCtlTermConnTalkingEv for T1B GC1: TermConnPassiveEv for T2B GC1: CallCtlTermConnBridgedEv	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(TA) returns ID25 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID26 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(T1) returns ID26 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(T2) returns ID26 (Connection of B).getUniqueID(null) returns returns ID26 (Connection of B).getUniqueID(T1) returns returns ID26 (Connection of B).getUniqueID(T2) returns returns ID26

Actions	Events	Call info
7. Shared Line Barge Scenario; All Observed; DN B is present on T1, T2 A calls B; B(T1) Answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: TermConnCreatedEv for T1B GC1: TermConnRingingEv for T1B GC1: CallCtlTermConnRingingEv for T1B GC1: TermConnCreatedEv for T2B GC1: TermConnRingingEv for T2B GC1: CallCtlTermConnRingingEv for T2B GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for T1B GC1: CallCtlTermConnTalkingEv for T1B GC1: TermConnPassiveEv for T2B GC1: CallCtlTermConnBridgedEv	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(TA) returns ID27 (Connection of B).getUniqueID(null) returns returns ID28 (Connection of B).getUniqueID(T1) returns returns ID28 (Connection of B).getUniqueID(T2) returns returns ID28

Actions	Events	Call info
B(T2) Presses Barge	GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for T2B GC2: TermConnCreatedEv for T1B GC2: CiscoCallChangedEv GC1: TermConnActiveEv for T2B GC1: CallCtlTermConnTalkingEv for T2B GC2: TermConnDroppedEv for T2B GC2: CallCtlTermConnDroppedEv for T2B GC2: TermConnDroppedEv for T2B GC2: CallCtlTermConnDroppedEv for T2B GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC2: CallInvalidEv GC2: CallObservationEndedEv	((CiscoConnection)(GC2: ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID29 (GC1: Connection of B).getUniqueID(T1) returns returns ID28 (GC1: Connection of B).getUniqueID(T2) returns returns ID30 (GC1: Connection of B).getUniqueID(null) is unpredictible (can be either of above two)

Actions	Events	Call info
8. Park/Unpark Scenario (All Observed) A calls B; B answers B does PARK C Unparks		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID31 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID32 ((CiscoConnection)(GC1: ConnCreatedEv for ParkDN).getConnection()).getUniqueID(null) throws PrivilegeVoilationException ((CiscoConnection)(GC2: ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID34 ((CiscoConnection)(GC1: ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID35

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for ParkDN GC1: CallCtlConnQueuedEv for ParkDN GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC2: CallActiveEv GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnInitiatedEv for C GC2: TermConnCreatedEv for TC GC2: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: TermConnCreatedEv for TC GC1: ConnDisconnectedEv for ParkDN GC1: CallCtlConnDisconnectedEv for ParkDN GC2: TermConnDroppedEv for TC	

Actions	Events	Call info
	GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC2: CallInvalidEv GC2: CallObservationEndedEv	
9. Conference Chaining Scenario; All Observed GC1: A calls B; B answers and adds C to conference GC3: C calls D; D answers and adds E to conference App sends GC1.conference(GC3) to chain two conferences	GC1 and GC3 are created as normal GC1 has connections for A, B and C(Held) GC3 has connections for C(Talking), D and E GC3: ConnCreatedEvent for cBridge-GC1 GC3: CiscoConferenceChainAddedEv GC3: ConnConnectedEvent for cBridge-GC1 GC3: CallCtlConnEstablishedEv for cBridge-GC1 GC3: TermConnDroppedEv for TC GC3: CallCtlTermConnDroppedEv for TC GC3: ConnDisconnectedEvent for C GC3: CallCtlConnDisconnectedEv C GC1: CallCtlTermConnTalkingEv for TC GC1: ConnCreatedEvent for cBridge-GC3 GC1: CiscoConferenceChainAddedEv GC1: ConnConnectedEvent for cBridge-GC3 GC3: CallCtlConnEstablishedEv for cBridge-GC3	(Connection for A).getUniqueID(null) returns ID37 (Connection for B).getUniqueID(null) returns ID38 (GC1: Connection for C).getUniqueID(null) returns ID39 (Connection for C).getUniqueID(null) returns ID40 (Connection for D).getUniqueID(null) returns ID41 (GC3: Connection for E).getUniqueID(null) returns ID42 ((CiscoConnection)(GC3: ConnCreatedEv for cBridge-GC1).getConnection()).getUniqueID(null) throws PrivilegeVoilationException ((CiscoConnection)(GC1: ConnCreatedEv for cBridge-GC3).getConnection()).getUniqueID(null) throws PrivilegeVoilationException (Connection for A).getUniqueID(null) returns ID37 (Connection for B).getUniqueID(null) returns ID38 (Connection for C).getUniqueID(null) returns ID39 (Connection for D).getUniqueID(null) returns ID41 (Connection for E).getUniqueID(null) returns ID42

Actions	Events	Call info
Application sends E.disconnect()	GC3: TermConnDroppedEv for TE GC3: CallCtlTermConnDroppedEv for TE GC3: ConnDisconnectedEvent for E GC3: CallCtlConnDisconnectedEv E GC1: ConnDisconnectedEvent for cBridge-GC3 GC1: CiscoConferenceChainRemovedEv GC1: CallCtlConnDisconnectedEv cBridge-GC3 GC3: CiscoCallChangedEv GC1: ConnCreatedEvent for D GC1: ConnConnectedEvent for D GC1: CallCtlConnEstablishedEv for D GC1: TermConnCreatedEvent for TD GC1: TermConnActiveEvent for TD GC1: CallCtlTermConnTalkingEv for TD GC3: ConnDisconnectedEvent for cBridge-GC1 GC3: CiscoConferenceChainRemovedEv GC3: CallCtlConnDisconnectedEv cBridge-GC31 GC3: TermConnDroppedEv for TD GC3: CallCtlTermConnDroppedEv for TD GC3: ConnDisconnectedEvent for D GC3: CallCtlConnDisconnectedEv D GC3: CallInvalidEvent GC3: CallObservationEndedEv	((CiscoConnection)(GC1: ConnCreatedEv for D).getConnection()).getUniqueID(null) returns ID43

Hunt Log Status for Phone Devices

In the following use cases A, B, C, and D are IP phones where A and B are a part of line group which is configured to hunt pilot HP. A is the first hunt member and it is logged out of the hunt group and B is the second hunt member and it is logged in to the hunt group on hunt pilot. For the following use cases the CiscoTermEvFilter. setHuntLogStatusChangedEvFilter() is set to true.

Call To Hunt Pilot where device is logged into hunt group

Action	Events	Call information
On A, huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus).	CiscoTermHuntLogStatusChangedEv on A	Ev.getHutLogStatus() = CiscoTerminal.DEVICE_HUNT_LOGGED_IN
C calls Hunt pilot HP.	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 CiscoHuntConnCreatedEv HP GC1 ConnInProgressEv HP GC1 CallCtlConnOfferedEv HP	CurrentCallingParty = C CurrentCalledParty = A
HP offers the call to A as it is the first hunt member and A starts ringing.	GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAltertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TermA GC1 TermConnRingingEv TermA GC1 CallCtlTermConnRingingEv TermA	
A answers the call	GC1 ConnConnectedEv HP1 GC1 CallCtlConnEstablishedEv HP1 GC1 ConnConnectedEv A GC1 CallCtlConnEstablishedEv A GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA	CurrentCallingParty = C CurrentCalledParty = A

Call To Hunt Pilot where device is logged out of the hunt group

Action	Events	Call information
On A, huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_OUT using the method setHuntLogStatus(int huntLogStatus) for the terminal to log in to the huntgroup.	CiscoTermHuntLogStatusChangedEv on A	Ev.getHuntLogStatus() = CiscoTerminal.DEVICE_HUNT_LOGGED_OUT
C calls Hunt pilot HP.	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC	CurrentCallingParty = C CurrentCalledParty = B
HP offers the call to B as A which is the first hunt member logged out of the hunt group and B is the second hunt member. B starts ringing.	GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 CiscoHuntConnCreatedEv HP GC1 ConnInProgressEv HP GC1 CallCtlConnOfferedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	
B answers the call	GC1 ConnConnectedEv HP1 GC1 CallCtlConnEstablishedEv HP1 GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB	CurrentCallingParty = C CurrentCalledParty = B

Updating the value of huntlogstatus on Unsupported device(Route Point/Spark Remote device/CTI Remote Device)

Action	Events	Call information
huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus) for the route point to log in to the huntgroup.		com.cisco.jtapi.MethodNotSupportedException:Operation not allowed

Updating the value of huntlogstatus on the terminal which is out of service

Action	Events	Call information
huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus) on D which is not in service		com.cisco.jtapi.InvalidStateException

Energywise Deep Sleep Mode

Scenario 1

JTAPI reports new reason“ENERGYWISE_POWER_SAVE_PLUS”in CiscoProvTerminalUnRegisteredEv and cause“CAUSE_ENERGYWISE_POWER_SAVE_PLUS”in CiscoTermOutOfServiceEv and CiscoAddrOutOfServiceEv to the application when a terminal/address unregisters from Cisco Unified CM due to deep sleep time.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of 'A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	
Terminal 'A' enters Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

Scenario 2

Terminal gets unregistered due to Deep Sleep mode and the user tries to manually register the terminal during the Deep Sleep time.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of 'A"	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	
Terminal 'A' goes to Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
A user tries to register the phone with Cisco Unified CM during deep sleep mode.		Cisco Unified IP 7900 Series phones do not re-register with the Cisco Unified CM during the Deep Sleep time. This is a limitation of the phone. Cisco Unified IP 9900 and 6900 Series phones register back with the Cisco Unified CM by pressing the select key on the phone.
Phone registers with the Cisco Unified CM after the Deep Sleep time expires.	[Term A] CiscoProvTerminalRegisteredEv [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	

Scenario 3

Shared line scenario. Two devices A (Cisco Unified IP Phones 7900 Series phone) and A' (CTI Port) are configured with the same line. Deep Seep mode is enabled on device A

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	

Description	Events	Information
Terminal A goes to Deep Sleep mode and gets unregistered. (Terminal A' remains in registered state.)	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

Scenario 4

Shared line scenario. Two devices A and A' (both are Cisco Unified IP Phones 7900 Series phones) that have are configured with the same line. Deep Sleep mode is enabled on A. Another device B calls the shared line after device A enters to the Deep Sleep mode.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	
Terminal A goes to deep sleep mode and gets unregistered. (Terminal A' remains in registered state.)	[Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER

Description	Events	Information
Another terminal B calls to the shared line DN.	GC1 CallActiveEv GC1 ConnCreatedEv [Addr B] GC1 ConnConnectedEv [Addr B] GC1 CallCtlConnInitiatedEv [Addr B] - GC1 ConnCreatedEv [Addr A'] GC1 ConnInProgressEv [Addr A'] GC1 CallCtlConnOfferedEv [Addr A'] - GC1 ConnConnectedEv [Addr A'] GC1 CallCtlConnEstablishedEv [Addr A'] GC1 TermConnActiveEv [Term A'] GC1 CallCtlTermConnTalkingEv [Term A'	

Scenario 5

Shared line scenario. Two device A (Cisco Unified IP Phones Series 9900/6900 phone) and A' (Cisco Unified IP Phones Series 9900/6900 phone) are configured with the same line. Deep Sleep mode is enabled on both devices.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	

Description	Events	Information
Terminal A and A' enters Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A'] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Term A'] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv [Addr A'] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
Deep Sleep mode power off time has expired and A and A' reregister to the Cisco Unified CM	Term A] CiscoProvTerminalRegisteredEv [Term A'] CiscoProvTerminalRegisteredEv [Term A] CiscoTermInServiceEv [Term A'] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Addr A'] CiscoAddrInServiceEv	

Scenario 6

Basic call scenario. Two devices A (CTI port) and B (Cisco Unified IP Phones 7900 Series phone) are configured on a Cisco Unified CM and Deep Sleep mode is enabled on B with power off time configured for 6:00 PM. A calls B at 5:55 pm and the call continues until 6:10 pm. The idle timer is set for 10 minutes.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and B	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term B] CiscoTermInServiceEv [Addr B] CiscoAddrInServiceEv	

Description	Events	Information
Terminal B calls A at 5:55 pm. A answers the call and goes to connected state.	GC1 CallActiveEv GC1 ConnCreatedEv [Addr B] GC1 ConnConnectedEv [Addr B] GC1 CallCtlConnInitiatedEv [Addr B] - GC1 ConnCreatedEv [Addr A] GC1 ConnInProgressEv [Addr A] GC1 CallCtlConnOfferedEv [Addr A] - GC1 ConnConnectedEv [Addr A] GC1 CallCtlConnEstablishedEv [Addr A] GC1 TermConnActiveEv [Term A] GC1 CallCtlTermConnTalkingEv [Term A]	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
At 6:00 pm Deep Sleep time is enabled but the call does not get dropped and remains active.		
The user disconnects the call from the phone at 6:10 PM and the idle timer (set for 10 minutes) starts.	GC1 TermConnDroppedEv [Term B] GC1 CallCtlTermConnDroppedEv [Term B] GC1 ConnDisconnectedEv [Addr B] GC1 CallCtlConnDisconnectedEv [Addr B] GC1 CallInvalidEv	
There is no action on phone A for the next 10 minutes. So at 6:20 pm, after the idle timer has expired, the terminal enters Deep Sleep mode and unregisters from the Cisco Unified CM.	[Term B] CiscoProvTerminalUnRegisteredEv [Term B] CiscoTermOutOfServiceEv [Addr B] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

External Call Control

You should assume that all devices in the following use cases are obsereved, unless explicilty stated otherwise in the use case description.

The first few use cases go through the full event series for the basic call setup. After the first three or four, the use cases leave this part out, as it is standard for most of the use cases. If you do not see the basic call event series at the beginning of a use case, you can assume that it was intended to have happened successfully before the first event in the use case.

The last column in the use cases, that specifies the call info for a various stage of the use case, will initially have the full method invocation to retrieve the call information, for example CiscoCall.getModifiedCallingParty(). After the first three or four uses cases, only the method name is specified, such as .getModifiedCallingParty(). You can assume that this is to be prefixed with CiscoCall unless explicitly stated otherwise, such as for the CiscoCallChangeEvs.

Use Cases for BasicCall

Basic Call Initiated From JTAPI / Phone

Configuration

Phone A, B are in cluster devices.

Procedure:

Application invokes connect() at A to call B, or physical phone for A dials the number for B.

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A	CiscoCall.getCurrentCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getCurrntCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall getCurrentCalledTerminal() = null
Connection of B created B starts ringing B Answers	GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B, CiscoCall.getCalledAddress() = B, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B

Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Called Party

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where called party get transformed to B1.

Procedure:

Application invokes connect() at A to call B.

Actions	Events	Call info
A initiates call to B Connection of A created,	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null

Actions	Events	Call info
Connection of B1 created B1 starts ringing B1 Answers	GC1-ConnCreatedEvent-B1 GC1-ConnInprogressEvent-B1 GC1-CallCtlConnOfferedEv-B1 GC1-ConnAlertingEvent-B1 GC1-CallCtlConnAlertingEv-B1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B1 GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getCurrentCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getCurrentCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Calling Party

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where calling party gets transformed to A1.

Procedure:

Application invokes connect() at A to call B.

Action	Events	Call info
A initiates call to B Connection of A created, Connection of B created B starts ringin	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall getCallingAddress() = A, CiscoCall getCurrentCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null
B Answers	GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B, CiscoCall.getCalledAddress() = B, CiscoCall.getCurrentCallingTerminal() = Terminal of A CiscoCall.getCurrentCalledTerminal() = Terminal of B

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Both Calling and Called Parties

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B

Action	Events	Call info
A initiates call to B Connection of A created, called party info set Connection of B1 created B1 starts ringing A gets CallStateChg For Ringback	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B1, CiscoCall getCalledAddress() = B1, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1,

Action	Events	Call info
B1 Answers	GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1

Called Party Redirects a Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 redirects the call in connected state.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 redirects the call to C Connection for C is created C rings B1 gets dropped A gets CallStateChg for Ringback	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1 CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = null
C Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = terminal of C

Called Party Which Has Transformed Calling and Called Parties Parks the Call and Receives a Park Reminder Call

Configuration

Phone A, B are in cluster devices. C is a park DN (also in cluster)

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 answers and then B1 parks the call and after park reversion timer expiry receives the reminder call.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 Parks the call Connection for C is created B1 gets park reminder B1 rings C gets dropped B1 Answers	… See use case 15.7.1.1.1 … GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnQueuedEv-C GC1-ConnCreatedEvent-B1 GC1-ConnInprogressEvent-B1 GC1-CallCtlConnOfferedEv-B1 GC1-ConnAlertingEvent-B1 GC1-CallCtlConnAlertingEv-B1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B1

Calling Party Parks the Call and Receives a Park Reminder Call After a Transformation From Called Party Translation Pattern

Configuration

Phone A, B are in cluster devices. C is a park DN

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 answers and then A parks the call and after park reversion timer expiry receives the reminder call.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) A Parks the call Connection for C is created A gets park reminder A rings	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnQueuedEv-C GC1-ConnCreatedEvent-A GC1-ConnInprogressEvent-A GC1-CallCtlConnOfferedEv-A GC1-ConnAlertingEvent-A GC1-CallCtlConnAlertingEv-A GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = A, .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = A, .getCalledAddress() = A, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of B1 .getCurrentCalledTerminal() = null
C gets dropped A Answers	GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = A, .getCalledAddress() = A, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of A

Caller Redirects a Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. A redirects the call in connected state.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) A redirects the call to C Connection for C is created C rings A gets dropped B1 gets CallStateChg for Ringback C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1 CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1.

Called Party Transfers the Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 consult transfer the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null

Actions	Events	Call info
Transfer starts Call Changes	GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv	.getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B1, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1

Actions	Events	Call info
Connection for C is created (GC1) C gets dropped (GC2) B1 gets dropped (GC1) B1 gets dropped (GC2) GC2 Invalid Transfer comple	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

Called Party Transfers the Call Which Has Transformed Calling and Called Parties to a DN Which Matches the Translation Pattern with Calling Party Transformation Defined

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

C matches the the translation pattern with calling party transformation to B2

Procedure:

Application invokes connect() at A to call B. B1 consult transfer the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C,

Actions	Events	Call info
Transfer starts Call Changes	GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv	.getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B2, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1 .getModifiedCallingAddress() = A1,

Actions	Events	Call info
Connection for C is created (GC1) C gets dropped (GC2) B1 gets dropped (GC1) B1 gets dropped(GC2) GC2 Invalid Transfer complete	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	.getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

Called Party with Transformed Calling and Called Parties Conferences a DN

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 consult conference the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv CG1-CallCtlTermConnHeldEv GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C

Actions	Events	Call info
Conference Starts B1 gets dropped (GC2)	GC1-CiscoConferenceStartEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC1-CiscoTermConnSelectChangedEv GC1-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv	Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B1, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1
Connection for C is created (GC1) C gets dropped (GC2) GC2 invalid Conferece Ends	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C * GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv-C GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoConferenceEndEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = null, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Called Party with Transformed Calling and Called Parties Conferences a DN Which Matches the Translation Pattern with Calling Party Transformation Defined

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

C has a translation pattern configured where calling party gets transformed to B2.

Procedure:

Application invokes connect() at A to call B. B1 consult conference the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv CG1-CallCtlTermConnHeldEv GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null

Actions	Events	Call info
Conference Starts B1 gets dropped (GC2)	GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoConferenceStartEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC1-CiscoTermConnSelectChangedEv GC1-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv	.getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B2, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C
Connection for C is created (GC1) C gets dropped (GC2) GC2 invalid Conferece Ends	GC1-ConnCreatedEvent-C * GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv-C GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoConferenceEndEv	FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1 .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = null, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

WildCard Routepoint Interaction (Behavior Change)

WildCard RoutePoint Redirects a Basic Incoming Call to IPPhone

Configuration

Phone A, B are in cluster devices. 4XXX is a wildcard routepoint

Service parameter “Use WildCard pattern in CTI Call Info” is set to true.

Procedure:

Application invokes connect() at A to call 4000. 4XXX redirects the call to B.

Actions	Events	Call info
A initiates call to 4000 Connection of A created, called party info set Connection of 4XXX created 4XXX Redirects to B Connection for B created B is ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-4XXX GC1-ConnInprogressEvent-4XXX GC1-CallCtlConnOfferedEv-4XXX GC1-ConnAlertingEvent-4XXX GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCalledAddress() = 4XXX, .getCurrentCalledAddress() = 4XXX .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4XXX,

Actions	Events	Call info
4XXX gets dropped B Answers	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4XXX GC1-CallCtlConnDisconnectedEv-4XXX GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getLastRedirectedAddress() = 4XXX, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCalledAddress() = 4XXX, .getLastRedirectedAddress() = 4XXX, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B

WildCard Routepoint Interaction (Original Behavior)

WildCard RoutePoint Redirects a Basic Incoming Call to IPPhone

Notes / Caveats

This configuration is not supported. This use case is only intended to show the call flow or events for the above use case with the Use WildCard pattern in CTI Call Info service parameter turned off. Applications should not count on this information to be correct, and to properly support Wildcard Routepoint scenarios, should look to adapting their applications so that they can support the new service parameter being enabled.

An important thing to note is that a connection is created for the dialed DN, 4000. This connection, as well as the connection of 4XXX is not dropped from the call until the redirect happens. This means that if a Wildcard DN is configured on a phone or device, you will see connections for the calling party, 4000, and 4XXX. This basic call will have three connections, which may confuse applications, which might believe it to be a conference call. CiscoCall.isConference() would still return false in this scenario. As stated in previous sections, this extra connection is created in error, and applications should not rely on this connection being there.

Configuration

Phone A, B are in cluster devices. 4XXX is a wildcard routepoint

Service parameter “Use WildCard pattern in CTI Call Info” is set to false / OFF.

Procedure:

Application invokes connect() at A to call 4000. 4XXX redirects the call to B.

Actions	Events	Call info
A initiates call to 4000 Connection of A created, called party info set Connection of 4000 created Connection of 4XXX created	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-4000 GC1-ConnInprogressEvent-4000 GC1-CallCtlConnOfferedEv-4000 GC1-ConnAlertingEvent-4000 GC1-ConnCreatedEvent-4XXX GC1-ConnInprogressEvent-4XXX GC1-CallCtlConnOfferedEv-4XXX	.getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCurrentCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCurrentCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCurrentCalledAddress() = 4000, .getCalledAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
4XXX Redirects to B Connection for B created B is ringing B Answers	GC1-ConnAlertingEvent-4XXX (note: 3 connections on the 2 party call.) GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4XXX GC1-CallCtlConnDisconnectedEv-4XXX GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4000 GC1-CallCtlConnDisconnectedEv-4000 GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCurrentCalledAddress() = 4000, .getCalledAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4000, .getLastRedirectedAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4000, .getLastRedirectedAddress() = 4000, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B

External Call Control Use Cases

External Call Control on Translation Pattern and CEPM Returns “continue”

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns continue and hence call will be presented to B1 (see use case “Basic Call initiated from JTAPI to the DN with Translation Pattern configured to transform called party” in the Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 925 topic).

External Call Control on Translation Pattern and CEPM Returns “divert”

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divert to C.

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCurrentCalledAddress() = BXXX, .getCalledAddress() = BXXX, .getLastRedirectedAddress() = BXXX .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCurrentCalledAddress() = C,

Actions	Events	Call info
C Answers	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getCalledAddress() = BXXX, .getLastRedirectedAddress() = BXXX .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = BXXX, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns <reject>

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B. The routing rule for B says “Reject”<reject>

CEPM returns reject.

A receives ConnFailedEvent (cause = CtiCallRejected), ConnDisconnectedEv (cause = normal), CallInvalidEvent (caue = Normal).

Actions	Events	Call info
A initiates call to B Connection of A created, CEPM Returns Reject	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnFailedEv-A GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-CallInvalidEvent GC1-CallObservationEndedEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null Cause = CtiCallRejected Cause = Normal

External Call Control on Translation Pattern and CEPM Returns “continue” with Modified Calling and Called Parties

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns continue with ModifiedCalling = “MA” and ModifiedCalled = “MB”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 925.)

Call Events:

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns continue with modified calling/called Connection of MBcreated MB starts ringing BAnswers	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-MB GC1-ConnInprogressEvent-MB GC1-CallCtlConnOfferedEv-MB GC1-ConnAlertingEvent-MB GC1-CallCtlConnAlertingEv-MB GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-MB GC1-ConnConnectedEvent-MB GC1-CallCtlConnEstablishedEv-MB GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getCurrentCallingAddress() = A .getModifiedCalledAddress() = MB, .getCurrentCalledAddess() = MB, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = MB, .getCalledAddress() = MB, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of MB

External Call Control on Translation Pattern and CEPM Returns “divert” with Modified Calling and Called Parties

Configuration:

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result:

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 925.)

Call Events:

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing C Answers	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = MB, .getCurrentCalledAddress() = C .getCalledAddress() = B1, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns “divert” with Modified Calling and Called Parties with resetCallHistory flag = resetLastHop

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”, resetCallHistory = “resetLastHop”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 925.)

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
C Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns "divert" with Modified Calling and Called Parties with resetCallHistory flag = resetAll

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = "C", with ModifiedCalling = "MA" and ModifiedCalled = "MB"

C has a userRule configured to DivertTo = "D" with ModifiedCalling = "MMA", ModifiedCalled = "MMB", resetCallHistory = "resetAll"

Call will be extended to "D"

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling CEPM Returns divert to D, modify Called/Calling Connection of D created D starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-D GC1-ConnInprogressEvent-D GC1-CallCtlConnOfferedEv-D GC1-ConnAlertingEvent-D GC1-CallCtlConnAlertingEv-D GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-D	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MMA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCurrentCalledAddress() = D, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null
D Answers	GC1-ConnConnectedEvent-D GC1-CallCtlConnEstablishedEv-D GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MMA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCalledAddress() = D, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of D

External Call Control on Translation Pattern and CEPM Returns <reject> and Service Parameter CTI Use Wildcard Pattern as calledPartyDN Is Set to False

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B says “Reject”<reject>

CEPM returns reject.

Jtapi throws platform exception to the application. A receives ConnFailedEvent (cause = CtiCallRejected), ConnDisconnectedEv (cause = normal), CallInvalidEvent (caue = Normal).

Actions	Events	Call info
A initiates call to B	GC1-CallActiveEvent	.getModifiedCallingAddress() = A,
Connection of A created,	GC1-ConnCreatedEvent-A	.getCallingAddress() = A,
CEPM Returns Reject	GC1-ConnConnectedEvent-A	.getModifiedCalledAddress() = “”,
	GC1-CallCtlConnInitiatedEv-A	.getCalledAddress() = “”,
	GC1-TermConnCreatedEvent	.getCurrentCallingTerminal() = Terminal of A.
	GC1-TermConnActiveEvent	.getCurrentCalledTerminal() = null
	GC1-CallCtlTermConnTalkingEv-A	.getModifiedCallingAddress() = A,
	GC1-CallCtlConnDialingEv-A	.getCallingAddress() = A,
	GC1-CallCtlConnEstablishedEv-A	.getModifiedCalledAddress() = “”,
	GC1-ConnFailedEv-A	.getCalledAddress() = “”,
	Jtapi throws Exception: PlatformException	.getCurrentCallingTerminal() = Terminal of A.
	GC1-TermConnDroppedEv	.getCurrentCalledTerminal() = null
	GC1-CallCtlTermConnDroppedEv	Exception info: Could not meet post conditions of connect()
	GC1-ConnDisconnectedEvent-A	Cause = CtiCallRejected
	GC1-CallCtlConnDisconnectedEv-A	Cause = Normal
	GC1-CallInvalidEvent	
	GC1-CallObservationEndedEv	

Transfer and External Call Control with Modified Calling and Called Parties

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX where External Call Contol is enabled.

Phone C and D does not match any translation pattern, and have no External Call Control defined.

Procedure:

Application invokes connect() at A to call B. CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”.

C initiate transfer to D and completes the transfer.

Result

Transfer is successfully completed

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = null, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = MB .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
C Answers C consult transfer to D Connection for C created (GC2) Connection for D created (GC2) D Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-CallCtlConnEstablishedEv-C GC2-ConnCreatedEvent-D GC2-ConnInprogressEvent-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEvent-D GC2-CallCtlConnAlertingEv-D GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-D GC2-CallCtlConnEstablishedEv-D GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = D, .getCalledAddress() = D, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = D, .getCalledAddress() = D,

Actions	Events	Call info
Transfer Starts D gets added to GC1	GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-D GC1-ConnConnectedEvent-D GC1-CallCtlConnEstablishedEv-D GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = terminal of D Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = C, OCall.getCallingAddress() = C, OCall.getModifiedCalledAddress() = D, OCall.getCalledAddress() = D, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of C OCall.getCurrentCalledTerminal() = terminal of D FCall.getModifiedCallingAddress() = MA, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = C, FCall.getCalledAddress() = C, FCall.getLastRedirectedAddress() = MB FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of C .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCurrentCalledAddress() = D, .getCalledAddress() = B1, .getLastRedirectedAddress() = C .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of D

Actions	Events	Call info
D gets dropped from GC2 C gets dropped from GC1 C gets dropped fromGC2 GC2 Invalid Transfer ends	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-D GC2-CallCtlConnDisconnectedEv-D GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	

Chaperone Use Cases

Call Is Redirected to a Hunt List of Chaperones and the Chaperone Enables Call Recording and Conferences in the Called Party

Configuration

A calls X, X's DN matches the translation pattern where External Call Control is enabled.

CEPM determines this call needs to have a chaperone's supervise. CEPM returns the permit decision with the obligation <divert>, destination HuntPilot B, which is a hunt pilot of chaperones, and a reason string "chaperone".

CUCM redirects the call to the hunt pilot B, and the chaperone C1 answers the call.

After talking to A briefly and discovered that A intended to talk to D, the chaperone C1 starts to establish a conference to D. C1 presses the conference softkey and dials D.

CUCM queries CEPM for the call, with calling user C1 with DN C1, and called user D with DN D.

CEPM returns the response with permit decision with <continue> call routing directive, since the policy server detects that the caller is the chaperone.

CUCM rings D's phone and D answers the call.

C1 presses the conference softkey again, and the conference is established.

The chaperone C1 presses the “record” softkey. This triggers the call recording being setup from C1’s IP phone to the recorder.

As one of the steps to establish recording calls to the recorder, two recording calls setup are first sent to the BIB of C1’s IP phone (INVITE for SIP phone and SCCP only the media message are involved). Note only one recording is shown in the picture.

As another step to establish the recording calls to the recorder, the two calls are then redirected to the recorder.

When the call recording is eablished successfully, the recording warning tone is playing to the C1’s phone. The recording warning tone is enabled by setting service parameter “Play Recording Notification Tone To Observed Target” to True.

After comfirming the call recording is established successfully, the chaperone reads an announcement to both A and D and informs them the call is being recorded.

A and D starts to talk under the supervision of the chaperone.

NOTE

Chaperones have limited abilities in what they can do on a call. The most obvious example is that they cannot put the call on hold, because they are required to be on the call at all times. To learn more about Chaperone limitations, please see the related sections of the External Call Control FFS.

Call Events:

Actions	Events	Call info
A initiates call to X Connection of A created, Hunt connection created (see Hunt List section) Connection of C1 created C1 starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-CiscoHuntConnCreatedEv-B GC1-ConnInProgressEv-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEv-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C1 GC1-ConnInprogressEvent-C1 GC1-CallCtlConnOfferedEv-C1 GC1-ConnAlertingEvent-C1 GC1-CallCtlConnAlertingEv-C1	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedParty() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X,

Actions	Events	Call info
C1 Answers C1 initiates conference Conference consult call to D , D answers	GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C1 GC1-ConnConnectedEvent-C1 GC1-CallCtlConnEstablishedEv-C1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlTermConnHeldEv-TermC1 GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEv-C1 GC2-CallCtlTermConnTalkingEv-TermC1 GC2-CallCtlConnDialingEv-C1 GC2-CallCtlConnEstablishedEv-C1 GC2-CiscoConnCreatedEv-D GC2-ConnInProgressEv-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEv-D	.getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C1 Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = .getCurrentCalledAddress() = .getCalledAddress() = .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = D .getCurrentCalledAddress() = D .getCalledAddress() = D

Actions	Events	Call info
Call 2 merges Conn for D created on GC1 Call 2 cleaned up Chaperone C1 starts recording Chaperone C1 tries to redirect the call	GC2-CallCtlConnAlertingEv-D GC2-ConnConnectedEv-D GC2-CallCtlConnEstablishedEv D CiscoCallChangedEv final call = GC1, consult call = GC2 GC1-ConnCreatedEv D GC1-ConnConnectedEv D GC1-CallCtlConnEstablishedEvD GC2-ConnDisconntedEv C1 GC2-ConnDisconntedEv D GC2-CallCtlConnDisconnectedEv C1 GC2-CallCtlConnDisconnectedEv D GC2-CallCtlTermConnDroppedEv C1 GC2-CallCtlTermConnDroppedEv D GC2-CallInvalidEv Normal recording events when recording is initiated on a conference call will be received. InvalidStateException : Did not meet pre conditions.	.getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = terminal of D

Call Is Redirected to a Hunt List of Chaperones and the Chaperone Conferences in the Called Party From Application

Configuration

A calls X, X's DN matches the translation pattern where External Call Control is enabled.

CUCM redirect the call to the hunt pilot B. Call is intercepted by the chaperone and the chaperone C1 answers the call.

After talking to A briefly and discovered that A intended to talk to D, the chaperone C1 starts to establish a conference to D. C1 initiates a consult call to D. A new global call id GC2 is created.

CUCM rings D's phone and D answers the call.

C1 invokes GC2.conference(GC1) from application.

At this step, request for establishing the conference would fail. Jtapi would throw InvalidStateException with the error code as "Call state not valid".

In order to establish a conference successfully, application must invoke the conference by passing the CI of the call in which chaperone is the controller as the primary CI. So in this case, if application invokes GC1.conference(GC2), it would be able to establish the conference successfully and if application invokes GC2.conference(GC1), Jtapi would throw an exception.

Also application can use CiscoConnection.isChaperone() API to determine controller is chaperone on which call.

Call Events:

Actions	Events	Call info
A initiates call to X Connection of A created, Hunt connection created (see Hunt List section) Connection of C1 created	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-CiscoHuntConnCreatedEv-B GC1-ConnInProgressEv-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEv-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C1 GC1-ConnInprogressEvent-C1 GC1-CallCtlConnOfferedEv-C1	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedParty() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X,

Actions	Events	Call info
C1 starts ringing C1 Answers C1 initiates conference	GC1-ConnAlertingEvent-C1 GC1-CallCtlConnAlertingEv-C1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C1 GC1-ConnConnectedEvent-C1 GC1-CallCtlConnEstablishedEv-C1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlTermConnHeldEv-TermC1 GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEv-C1 GC2-CallCtlTermConnTalkingEv-TermC1 GC2-CallCtlConnDialingEv-C1 GC2-CallCtlConnEstablishedEv-C1	.getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C1 Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = .getCurrentCalledAddress() = .getCalledAddress() = .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1

Actions	Events	Call info
Conference consult call to D , D answers C1 completes the conference by invoking GC2.conference(GC1) from application. C1 tries to complete the conference by invoking GC1.conference(GC2_from application	GC2-CiscoConnCreatedEv-D GC2-ConnInProgressEv-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEv-D GC2-CallCtlConnAlertingEv-D GC2-ConnConnectedEv-D GC2-CallCtlConnEstablishedEv D InvalidStateException : Call state not valid. CiscoCallChangedEv final call = GC1, consult call = GC2 GC1-ConnCreatedEv D GC1-ConnConnectedEv D GC1-CallCtlConnEstablishedEv D GC2-ConnDisconntedEv C1 GC2-ConnDisconntedEv D GC2-CallCtlConnDisconnectedEv C1 GC2-CallCtlConnDisconnectedEv D GC2-CallCtlTermConnDroppedEv C1 GC2-CallCtlTermConnDroppedEv D GC2-CallInvalidEv	.getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = D .getCurrentCalledAddress() = D .getCalledAddress() = D .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = terminal of

Extension Mobility Cross Cluster

Actions	Events	Call info
1. User1 has a device profile configured with DN A in cluster1. This profile is included in the control list of application. User1 goes to a visiting cluster and EM login to a device TERMA. Device registers to cluster1	Events to provider observer CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA	CiscoTerminal.getLoginType() returns CiscoTerminal.NO_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN CiscoTerminal.getLoginType() returns CiscoTerminal.VISITOR_LOGIN

Actions	Events	Call info
User1 logs off from the Device	CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT CiscoTerminal.getLoginType() returns CiscoTerminal.NO_LOGIN
2. User1 has a device profile configured with DN A in cluster1. This device profile is included in the control list of application. User1 EM into a device TERMA on cluster1. Device re-registers with DN A. The device TERMA is not in application control listapplication control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN CiscoTerminal.getLoginType() returns CiscoTerminal.NATIVE_LOGIN
User1 log off from the device. Device re-registers to cluster1 with default DN.	CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
3. EM into a controlled Device: User1 has a device profile configured with DN A in cluster1. This device profile is included in the control list of application. User1 EM into a device TERMA on cluster1. TERMA with default DN X is in application control list. Device re-registers with DN A.	CiscoAddrRemovedEv X CiscoAddrCreatedEv A	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN
User1 logs out of the device. Device Unregister, Device and line out of service Device Register to CM with default DN X.	CiscoAddrRemovedEv A CiscoAddrCreatedEv X	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
4. Application uses user1 userid. Device profile agent1 is added to application control list after application is started User EMs into a device TERMA and gets the device profile. Agent1 device profile is removed from application control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_REMOVE getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE

Actions	Events	Call info
5. EMCC scenario resulting in CiscoAddrAddedToTerminalEv and CiscoAddrRemovedFromTerminalEv Cluster1 has application with Terminal TermA (address A) in control list. User1 is a device profile which is configured with line A is included in app control list. User goes to a visiting cluster and logs into a device (TermX, Addr X). TermX registers with cluster1 with address A User logs out of device TermX	CiscoTermCreatedEv TERMX CiscoAddrAddedToTerminalEv AddrA CiscoAddrRemovedFromTerminalEv AddA CiscoTermRemovedEv TERMx	getCiscoCause() returns CiscoProvEv.CAUSE_EM getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
6. Device profile Agent1 with DN A is logged into a device TERMA. User1 opens provider and then adds the profile Agent1 to the control list through the admin pages. User1 then removes the profile from the control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_ADD getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_ADD getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE
7. Device profile Agent1 is not in the applications control list but it is there as a controlled profiles for extension mobility for user1. User1 opens provider and logs into terminal TERMA with profile agent1 and the same user id with which it had opened the provider. User1 logs out of the device	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
8. Device profile Agent1 (DN A) is in the applications control list with user as user1. User1 opens the provider and does an EM login into TERMA with profile as agent1. TERMA is not in control list. User1 logs out of TERMA.	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT

Actions	Events	Call info
9. Device profile Agent1 (DN A) is in the applications control list with user as user1. User1 opens the provider and does an EM login into TERMA with profile as agent1. TERMA is in control list with default DN as X. User1 logs out of TERMA.	CiscoAddrRemovedEv X CiscoAddrCreatedEv A CiscoAddrRemovedEv A CiscoAddrCreatedEv X	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT

End to End Session ID for Calls

Session ID in a Basic Call Scenario

Application has already opened provider and observing TermA and TermB

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA	
	GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A	((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnA) = A's LocalUUID

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB	
	GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnA) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnB) = A's LocalUUID
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

SessionID for a Basic Call involving SIP Endpoints

Application has already opened provider and observing SIP terminals TermA and TermB

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = null ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = null ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = A's local UUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = B's localUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = null ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID

Action	Events	Call information
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

SessionIDs for Calls Involving Shared Lines and Hold Resume

B and B are lines shared on two terminals. Application has opened provider and observed A, B and B

Action	Events	Call information
Application makes a call between TermA and TermB		((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = B's or (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID

Action	Events	Call information
	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv	

Action	Events	Call information
	TermB GC1 TermConnCreatedEv TermB' GC1 TermConnRingingEv TermB' GC1 CallCtlTermConnRingingEv TermB'	
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv TermB GC1 TermConnPassiveEv TermB' GC1 CallCtlTermConnBridgedEv TermB'	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

Action	Events	Call information
B puts the call on hold	GC1 TermConNActiveEv TermB GC1 CallCtlTermConnHeldState TermB GC1 TermConnActiveEv termB' GC1 CallCtlTermConnHeldState TermB'	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB') = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB') = A's localUUID

Action	Events	Call information
B' resumes the call	GC1 TermConnActiveEv TermB' GC1 CallCtlTermConnTalkingEv TermB' GC1 TermConnPassiveEv TermB GC1 CallCtlTermConnBridgedEv TermB	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB') = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB') = A's localUUID

SessionID when Call is Redirected to a Third Party

Application has already opened provider and observed A,B and C and establishes a call between A and B.

Action	Events	Call information
A calls B and B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

Action	Events	Call information
Application redirects the call from B to C	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 TerConnDroppedEv TermB CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnC) = C's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnC) = A's LocalUUID CallInfo : CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(C)).getLocalUUID(termConnC) = C's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = C's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(C)).getPeerUUID(termConnC) = A's localUUID

Forced Authorization and Customer Matter Codes

Scenario One

The application controls A and B; B requires a forced authorization code (FAC) to extend the call.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. FAC_REQUIRED
Application enters additional digits by using CiscoConnection.addToAddress.	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv BTermConnRingingEv B CallCtlTermConnRingingEv B ConnConnectedEv B CallCtlConnEstablishedEv B
B answers the call.	TermConnActiveEv B

Scenario Two

The application controls A and B; B requires both an FAC and a CMC (client matter code) to extend the call.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. FAC_CMC_REQUIRED
Application enters FAC code digits with # termination by using CiscoConnection.addToAddress within the T302 timer.	NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. CMC_REQUIRED
Application enters CMC code digits with # terminated by using CiscoConnection.addToAddress within T302 timer.	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B

Action	Event
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B

Scenario Three

The application controls A and B;

B requires a CMC, and the application enters an invalid code.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEvToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. CMC_REQUIRED

Action	Event
The application enters the incorrect CMC digits (# terminated) by using CiscoConnection.addToAddress within the T302 timer limit. The application receives reorder tone.	NEW META EVENT_____META_CALL_PROGRESS ConnFailedEv A CallCtlConnFailedEv A getCiscoCause () = CiscoCallEv.FAC_CMC NEW META EVENT_____META_CALL_ENDING TermConnDroppedEv CallCtlTermConnDropped ConnDisconnectedEv CallCtlConnDisconnectedEv CallInvalidEv CallObservationEndedEv

Scenario Four

The application controls both A and B; A calls B; B redirects the call to C, which needs both an FAC and a CMC.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW METAEVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv SEPB TermConnRingingEv SEPB CallCtlTermConnRingingEv SEPB ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv SEPB CallCtlTermConnTalkingEv SEPB

Action	Event
B issues a redirect request to C and passes an FAC and a CMC code.	NEW META EVENT_____META_CALL_REMOVING_PARTY TermConnDroppedEv SEPB CallCtlTermConnDroppedEv SEPB Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnDisconnectedEv B Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnInProgressEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv A Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR

Scenario Five

Application controls the device Route Point (RP) and registers the RP.

A and B are PNO and within the Cisco Unified Communications Manager cluster.

Action	Event	Fields
Call arrives at RP	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A)
Application invokes selectRoute(routeselected[], callingsearchspace, modifiyingcallingnumber[], preferredOriginalCdNumber[], preferredOriginalCdOption[], facCode[], cmcCode[]) where routeSelected[] = BcallingSearchSpace = CiscoRouteSession.DEFAULT_SEARCH_SPACEmodifyingCgNumber = null, preferredOriginalCdNumber = null, preferredOriginalCdOption = CiscoRouteSession.DONOT_RESET_ORIGINALCALLED, facCode[] = “facCode for B”cmcCode[] = “cmcCode for B”	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A) getRouteUsed () = B
Application invokes endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_ENDgetRouteAddress () = RP

Hairpin Support

S.No.	Pre-Condition	Use Case	Expected Behavior	Result
1	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C. Gateway does not pass new party information to each other. There will be no transfer start and end events as transfer controller is not a controlled device.	A calls B via gateway. B transfers call to C via gateway. B completes the transfer and goes out of scenario. Now IP Phones A and C are connected	At A: It is connected to B. A’s type is CiscoAddress.Internal B’s type is CiscoAddress.External At C: It is connected to B. C’s type is CiscoAddress.Internal B’s type is CiscoAddress.External	A and C are connected.
2	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C. Gateway is able to pass new party information to each other.	A calls B via gateway. B transfers call to C via gateway. B completes the transfer and goes out of scenario. Now IP Phones A and C are connected.	At A: It is connected to C. A’s type is CiscoAddress.Internal C’s type is CiscoAddress.External At C: It is connected to A. C’s type is CiscoAddress.Internal A’s type is CiscoAddress.External	A and C are connected.

S.No.	Pre-Condition	Use Case	Expected Behavior	Result
3	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A and B.	A calls B. B does a conference call to C via gateway. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
4	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A, B and C.	A calls B. B does a conference call to C via gateway. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
5	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway is able to pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
6	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway does not pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal D: CiscoAddress.External	A, B and C are in conference call.
7	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C.	A calls B via gateway. B redirects call to C. Now IP Phones A and C are connected.	At A: It is connected to C. A's type is CiscoAddress.Internal C's type is CiscoAddress.External At C: It is connected to A. C's type is CiscoAddress.Internal A's type is CiscoAddress.External	A and C are connected.

Half Duplex Media

RTP Event at A and B.

Action	RTP Events	Check Interface
A calls B, B answers the call.	A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false
B puts Call on hold	A – CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv B – CiscoRTPInputStoppredEv CiscoRTPOutputStoppedEv A-CiscoRTPInputStartedEv	Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns True
B Retrieves the Call	A- CiscoRTPInputStoppredEv A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() returns True Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false

Hunt List

Configuration

- HuntList feature is enabled for all use cases, unless otherwise indicated
- HuntList pilot1 : 2000
- HuntList1 LineGroup Member : 3001, 3002, 3003
- HuntList pilot2: 4000
- HuntList2 LineGroup Member : 5001, 5002, 5003

Cisco Hunt Address mentioned in the call models below indicates that CiscoAddress.getType() returns CiscoAddress.HUNT_PILOT.

Scenario 1

Scenario	Result
A (1000) calls Hunt Pilot B (2000), Application is observing only A. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA CallingParty = A, current calling = A Called Party = null, current called = null Lrp = null GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000 type = CiscoAddress.HUNT_PILOT

CurrentCalledParty = 2000 type = CiscoAddress.HUNT_PILOT

LastRedirectingParty = Null

Current called display name = 2000Name.

Scenario 2

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001); application is observing A. GC1 is the GCID of the call. C(3001) answers the call	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA CallingParty = A, current calling = A Called Party = null, current called = null Lrp = null GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Current called party display name = 3001Name. Called party display name changes to the display name of the hunt pilot member that answered the call.

Scenario 3

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001). Application is observing C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC:

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 4

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null \

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 5

Scenario	Result
A (1000) calls Hunt Pilot (B or 2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1:CallCtlTermConnTalkingEv TermC

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 6

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001). Application is observing A and C. GC1 is the GCID of the call. A redirects the call to another Hunt Pilot D(4000)	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1:CallCtlTermConnTalkingEv TermC GC1: CiscoHuntConnCreatedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnDroppedEv TA GC1: CallCtlTermConnDroppedEv TA getCallControlCause () = CAUSE_REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause () = CAUSE_REDIRECTED

JTAPI CallInfo

CallingParty = 2000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 7

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call. A redirects the call to D(4000). E(5001) answers the call	….. ….. GC1: CiscoHuntConnCreatedEv D GC1: TermConnDroppedEv TermA GC1: CallCtlTermConnDroppedEv TermA getCallControlCause () = CAUSE_REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause () = CAUSE_REDIRECTED GC1: ConnCreatedEv E GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 8

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001). Application is observing A, E and C. GC1 is the GCID of the call. A redirects the call to D(4000). E(5001) answers the call	….. ….. GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E getCallControlCause () = CAUSE_REDIRECTED GC1: CiscoHuntConnCreatedEv D GC1: TermConnDroppedEv TermA GC1: CallCtlTermConnDroppedEv TermA GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A GC1: CallCtlConnEstablishedEv E

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 9

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001) and D (3002). Application is observing A, C and D. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: CallCtlConnAlertingEv C GC1: CallCtlTermConnCreatedEv TermC GC1: CallCtlTermConnRingingEv TermC GC1: CallCtlConnAlertingEv D GC1: CallCtlTermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 10

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001) and D (3002). Application is observing A, C and D. GC1 is the GCID of the call. D answers the call	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: CallCtlConnAlertingEv C GC1: CallCtlTermConnCreatedEv TermC GC1: CallCtlTermConnRingingEv TermC GC1: CallCtlConnAlertingEv D GC1: CallCtlTermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD GC1: CallCtlTermConnTalkingEv TermD GC1: CallCtlTermConnDroppedEv TermC GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 11

Scenario	Result
A (1000) calls Hunt Pilot (B or 2000), call is offered at C (3001) and D (3002). Application is observing A and D. GC1 is the GCID of the call. C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnCreatedEv D GC1: ConnOfferedEv D …. GC1: TermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: CallCtlTermConnDroppedEv TermD GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 12

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) and is answered. A consults with D (4000) and call is offered at E(5001). A completes the conference. Application is observing A. Initially connection is created to an address with DN = B type = UNKNOWN GC1 is the GCID of the final call. GC2 is the consult call C answers the call E answers the call Conference is completed	

Scenario	Result
	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B-U GC1: ConnInProgressEv B-U GC1: CallCtlConnOfferedEv B-U GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnDisconnectedEv B-U GC1: CallCtlConnDisconnectedEv B-U GC1: ConnCreatedEv C GC1: ConnOfferedEv C …. GC1: CallCtlConnEstablishedEv C GC1: CallCtlTermConnHeldEv TermA GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv A ….. GC2: CallCtlTermConnTalkingEv TermA GC2: CallCtlConnDialingEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoHuntConnCreatedEv D GC2: ConnInProgressEv D GC2: CallCtlConnOfferedEv D GC2: ConnAlertingEv D

Scenario	Result
	GC2: CallCtlConnAlertingEv D GC2: ConnCreatedEv E GC2: ConnConnectedEv E GC2: CallCtlConnEstablishedEv E GC2: ConnConnectedEv D GC2: CallCtlConnEstablishedEv D CiscoCallChangedEv final call –GC1, consult call = GC2 GC1: CiscoHuntConnCreatedEv B GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: CiscoHuntConnCreatedEv E GC1: ConnCreatedEv E GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC2: ConnDisconntedEv B GC2: ConnDisconntedEv C GC2: CallCtlConnDisconnectedEv C GC2: ConnDisconntedEv D GC2: ConnDisconntedEv E GC2: CallCtlConnDisconnectedEv E GC2: CallCtlTermConnDroppedEv TermA .. GC2: ConnDisconntedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = [No guaranted for conference scenario]

CalledParty = 2000

CurrentCalledParty = [No guaranted for conference scenario]

LastRedirectingParty = 1000

Scenario 13

Transfer to a line group member.

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A, B and C. GC1 is the GCID of the final call. GC2 is the consult call B answers the call B consults to Hunt pilot C answers the call Transfer is completed	

Scenario	Result
	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv B GC1: ConnOfferedEv B … GC1: TermConnRingingEv TermB GC1: TermConnTalkingEv TermB GC1: CallCtlTermConnHeldEv TermB GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv B ….. GC2: CallCtlTermConnTalkingEv TermB GC2: CiscoHuntConnCreatedEv P GC2: ConnInProgressEv P GC2: CallCtlConnOfferedEv P GC2: ConnCreatedEv C GC2: ConnOfferedEv C GC2: TermConnRingingEv TermC GC2: ConnConnectedEv P GC2: CallCtlConnEstablishedEv P GC2: CiscoHuntConnCreatedEv P GC2: ConnInProgressEv P GC2: CallCtlConnOfferedEv P GC2: ConnAlertingEv P GC2: CallCtlConnAlertingEv P GC2: ConnDisconnectedEv P GC2: CallCtlConnDisconnectedEv P

Scenario	Result
	GC2: ConnConnectedEv P GC2: CallCtlConnEstablishedEv P GC2: ConnConnectedEv C GC2: CallCtlConnEstablishedEv C GC2: TermConnTalkingEv TermC GC1: CiscoHuntConnCreatedEv P GC1: ConnInProgressEv P GC1: CallCtlConnOfferedEv P GC1: ConnAlertingEv P GC1: CallCtlConnAlertingEv P CiscoCallChangedEv final call –GC1, consult call = GC2 GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnTalkingEv TC GC1: ConnConnectedEv P GC1: CallCtlConnEstablishedEv P GC2: ConnDisconntedEv P GC2: CallCtlConnDisconnectedEv P GC2: ConnDisconntedEv B GC2: CallCtlConnDisconnectedEv B … …. GC2: ConnDisconntedEv C GC2: CallCtlConnDisconnectedEv C .. GC2: CallInvalidEv GC1: ConnDisconntedEv B GC1: CallCtlConnDisconnectedEv B GC1: TermConnDroppedEv TB GC1: CallCtlTermConnDroppedEv TB

Scenario 14

Pickup from line group

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. C and D are in the same pickup group. D picks up the call ringing at C. GC2 is the initial call at D. A and D are connected on GC1 D goes off-hook and answers call from C.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC2: CallActiveEv GC2: ConnCreatedEv D GC2: ConnConnectedEv D GC2: CallCtlConnInitiatedEv D GC2: TermConnCreatedEv TD GC2: TermConnActiveEv TD GC2: CallCtlTermConnTalkingEv TD GC2: CiscoCallChangedEv GC2->GC1 GC1: ConnCreatedEv D GC1: ConnConnectedEv D GC1: CallCtlConnInitiatedEv D GC1: TermConnCreatedEv TD GC1: TermConnActiveEv TD GC1: CallCtlTermConnTalkingEv TD GC2: TermConnDroppedEv TD GC2: CallCtlTermConnDroppedEv TD GC2: ConnDisconnectedEv D

Scenario	Result
	GC2: CallCtlConnDisconnectedEv D GC2: CallInvalidEv GC1: TermConnDroppedEv TC GC1: TermConnTermConnDroppedEv TC GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B

Note Note: For this scenario, if the pickup is done from the address of the hunt member that is currently ringing with Auto Pickup disabled, then getCiscoHuntConnection() returns the connection to the hunt pilot. If the pickup is done from an address that is in the pickup group but is not the current ringing terminal, then getCiscoHuntConnection() returns null. If Auto Pickup is enabled, then getCiscoHuntConnection() always returns null after the call is picked up (it does not matter whether the pickup is done from the ringing terminal or from another address in the pickup group). This is true for Pickup, Group Pickup, Other Pickup, and Directed Call Pickup.

Scenario 15

Gpickup a ringing hunt list member.

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. D picks up the call ringing at C. GC2 is the initial call at D. A and D are connected on GC1	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B

Scenario	Result
D goes off-hook and dials the pickup number Z.	

Scenario	Result
	GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC2: CallActiveEv GC2: ConnCreatedEv D GC2: ConnConnectedEv D GC2: CallCtlConnInitiatedEv D GC2: TermConnCreatedEv TD GC2: TermConnActiveEv TD GC2: CallCtlTermConnTalkingEv TD GC2: CallCtlConnDialingEv D GC2: ConnCretatedEv Z GC2: ConnInProgressEv Z GC2: CallCtlConnOfferedEv Z GC2: CallCtlConnEstablishedEv D GC2: CiscoCallChangedEv GC2->GC1 GC1: ConnCreatedEv D GC1: ConnCreatedEv Z GC1: ConnConnectedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnCreatedEv TD GC1: TermConnActiveEv TD GC1: CallCtlTermConnTalkingEv TD GC1: ConnInProgressEv Z GC1: CallCtlConnOfferedEv Z GC2: ConnDisconnectedEv Z GC2: CallCtlConnDisconnectedEv Z GC2: TermConnDroppedEv TD GC2: CallCtlTermConnDroppedEv TD GC2: ConnDisconnectedEv D

Scenario	Result
	GC2: CallCtlConnDisconnectedEv D GC2: CallInvalidEv GC1: ConnDisconnectedEv Z GC1: CallCtlConnDisconnectedEv Z GC1: TermConnDroppedEv TC GC1: TermConnTermConnDroppedEv TC GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B

Note For this scenario, if the pickup is done from the address of the hunt member that is currently ringing with Auto Pickup disabled, getCiscoHuntConnection() returns the connection to the hunt pilot. If the pickup is done from an address that is in the pickup group but is not the current ringing terminal, getCiscoHuntConnection() returns null. If the Auto Pickup is enabled, getCiscoHuntConnection() always returns null after the call is picked up (it does not matter whether the pickup is done from the ringing terminal or from another address in the pickup group). This is true for Pickup, Group Pickup, Other Pickup, and Directed Call Pickup.

Scenario 16

Redirect by a hunt member:

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. C redirects the call to D. D is not a member.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnEstablishedEv B GC1: ConnAlertingEv B

Scenario	Result
C answers the call. Application redirects the call from C to D	GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: CallCtlEstablishedEv C GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D getCallControlCause() = CAUSE.REDIRECTED GC1: CallCtlConnDisconnectedEv B GC1: CallCtlConnDisconnected C GC1: TermConnDisconnEv C GC1: CallCtlTermConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called D LRP B type CiscoAdress.UNKNOWN

Scenario 17

Calls Moving Between Members

When call is moving between hunt members, the call could go to invalid state.

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call is offered at D. Application is observing C and D Call moves to D.	GC1: CallActiveEv … GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: CallCtlConnEstablishedEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv TC getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEvTC GC1: CallCtlConnDisconnectedEv TC getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called D LRP = null

Scenario 18

Not All Members Are Observed

If all members are not observed call could go to invalid state when moving between hunt members

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E where it is answered. C, D, E and F are the members. Application is observing C and E.	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC
Call moves to D (not unobserved).	GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause() = CAUSE.REDIRECTED GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: CallInvalidEv GC1: CallActiveEv

Scenario	Result
Call moves from D to E.	GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE
E answers the call.	GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC1: TermConnActiveEv TE GC1: CallCtlTermConnTalkingEv TE Call info: Current calling A Current Called E LRP = null

Scenario 19

Not All Members Are Observed, but Calling Party Is Observed

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E where it is answered. C, D, E and F are the members. Application is observing A, C and E. Call moves to D (not unobserved). Call moves from D to E. E answers the call.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1; ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE

Scenario	Result
	GC1: CallCtlTermConnRingingEv TE GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC1: TermConnActiveEv TE GC1: CallCtlTermConnTalkingEv TE Call info: Current calling A Current Called E LRP = null

Scenario 20

Calling and All Hunt List Members Are Observed; the Call Is Not Answered and Goes to Hunt No Answer Forward Destination

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E. The call goes to hunt no answer forward destination F which is observed. C, D, E and F are the members.Application is observing A, C, D, E and F. Call moves to D. Call moves from D to E. Call moves to F.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1; ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C

Scenario	Result
	GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE GC1: TermConnDroppedEv TD GC1: CallCtlTermConnDroppedEv TD getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv F GC1: ConnInProgressEv F GC1: CallCtlConnOfferedEv F GC1: ConnAlertingEv F GC1: CallCtlConnAlertingEv F GC1: TermConnRingingEv TF GC1: CallCtlTermConnRingingEv TF GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B GC1: TermConnDroppedEv TE GC1: CallCtlTermConnDroppedEv TE getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv E GC1: CallCtlConnDisconnectedEv E getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called F

Scenario	Result
	LRP = null

Scenario 21

Forward Hunt No Answer to Another Hunt Pilot

Scenario	Result
A (1000) calls (GC1) Hunt Pilot HP1 (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E. The call goes to hunt no answer forward destination F which is observed. C, D, E are the members . HP2 is the forward hunt no answer destination. H, L are its members. All parties are observed.	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv HP1 GC1; ConnInProgressEv HP1 GC1: CallCtlConnOfferedEv HP1 GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv HP1 GC1: CallCtlConnEstablishedEv HP1

Scenario	Result
Call moves to D Call moves from D to E.	GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE GC1: TermConnDroppedEv TD GC1: CallCtlTermConnDroppedEv TD getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D getCallControlCause() = CAUSE.REDIRECTED

Scenario	Result
Call goes t HP2, call is offered at H Call moves from H to L	GC1: ConnCreatedEv H GC1: ConnInProgressEv H GC1: CallCtlConnOfferedEv H GC1: CiscoHuntConnCreatedEv HP2 GC1: ConnAlertingEv H GC1: CallCtlConnAlertingEv H GC1: TermConnRingingEv TH GC1: CallCtlTermConnRingingEv TH GC1: ConnConnectedEv HP2 GC1: CallCtlConnEstablishedEv HP2 GC1: ConnCreatedEv L GC1: ConnInProgressEv L GC1: CallCtlConnOfferedEv L GC1: ConnAlertingEv L GC1: CallCtlConnAlertingEv L GC1: TermConnRingingEv TL GC1: CallCtlTermConnRingingEv TL GC1: ConnDisconnectedEv HP1 GC1: CallCtlConnDisconnectedEv HP1 GC1: TermConnDroppedEv TH GC1: CallCtlTermConnDroppedEv TH getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv H GC1: CallCtlConnDisconnectedEv H getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called F LRP = null

Scenario 22

Consult Transfer by a Member to Another Hunt Pilot

Scenario	Result
A (1000) calls (GC1) Hunt Pilot HP1 (2000), call is offered at C (3001). C answers the call and consult to HP2. L in HP2 is ringing. C completes the transfer. C and L are observed C answers the call C consults with HP2 (GC2)	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv HP1 GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv HP1 GC1: CallCtlConnEstablishedEv HP1 GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnActiveEv TC GC1: CallCtlTermConnTalkingEv TC GC1: CiscoTermConnSelectChangedEv TC GC1: CallCtlTermConnHeldEv TC GC2: ConsultCallActive GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnInitiatedEv C GC2: TermConnCreatedEv C GC2: TermConnActiveEv C GC2: CallCtlTermConnTalkingEv TC GC2: CallCtlConnDialingEv TC GC2: CallCtlConnEstablishedEv TC GC2: CiscoHuntConnCreatedEv HP2 GC2: ConnInProgressEv HP2 GC2: CallCtlConnOfferedEv HP2

Scenario	Result
Call is offered to L C completes the transfer	GC2: ConnCreatedEv L GC2: ConnInProgressEv L GC2: CallCtlConnOfferedEv L GC2: ConnAlertingEv L GC2: CallCtlConnAlertingEv L GC2: TermConnRingingEv TL GC2: CallCtlTermConnRingingEv TL GC2: ConnConnectedEv HP2 GC2: CallCtlConnEstablishedEv HP2 GC2: CiscoCallChangedEv GC1: ConnCreatedEv L GC1: ConnAlertingEv L GC1: CallCtlConnAlertingEv L GC1: TermConnCreatedEv TL GC1: TermConnRingingEv TL GC1: CallCtlTermConnRingingEv TL GC2: ConnDisconnectedEv HP2 GC2: CallCtlConnDisconnectedEv HP2 GC2: TermConnDroppedEv TL GC2: CallCtlTermConnDroppedEv TL GC2: ConnDisconnectedEv L GC2: CallCtlConnDisconnectedEv L GC2: TermConnDroppedEv C GC2: CallCtlTermConnDroppedEv C GC2: ConnDisconnectedEv C GC2: CallCtlConnDisconnectedEv C GC1: ConnDisconnectedEv HP1 GC1: CallCtlConnDisconnectedEv HP1 GC2: CallInvalidEv GC1: CiscoHuntConnCreatedEv HP2 GC2: ConnConnectedEv HP2 GC2: CallCtlConnEstablishedEv HP2

Scenario	Result
L answers the call	GC1: TermConnDroppedEv C GC1: CallCtlTermConnDroppedEv C GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C GC1: ConnConnectedEv L GC1: CallCtlConnEstablishedEv L GC1: TermConnActiveEv L GC1: CallCtlTermConnTalkingEv TL

The following call scenarios are generally un-supported and applications are encouraged to enable the huntlist feature and adapt to the event flows described above.

Following are the expected events when the feature is **disabled**.

Scenario 23

Hunt list feature is disabled.

Basic call to hunt pilot

Scenario	Result
A (1000) calls Hunt Pilot P (2000), call is offered at C (3001) and is answered. Application is observing A. GCID is the call is GC1. C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv P GC1: ConnOfferedEv P … GC1: ConnAlertingEv P GC1: ConnConnected P GC1: CallCtlConnEstablishedEv A

bvHunt list feature is disabled

Scenario 24

Consult – Transfer Scenario

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A and B. GC1 is the GCID of the final call. GC2 is the consult call B answers the call B consults to Hunt pilot C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv B GC1: ConnOfferedEv B … GC1: TermConnRingingEv TermB GC1: TermConnTalkingEv TermB GC1: CallCtlTermConnHeldEv TermB GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv B ….. GC2: CallCtlTermConnTalkingEv TermB GC2: ConnCreatedEv P .. .. GC2: ConnAlertingEv P GC2: CallCtlConnEstablishedEv P

Scenario	Result
Transfer is completed	CiscoTransferStartEv final call –GC1, consult call = GC2 GC1: ConnCreatedEv P CiscoCallChangedEv GC2 = >GC1 GC2: ConnDisconnectedEv P GC2: ConnDisconntedEv B GC2: CallCtlConnDisconnectedEv B … …. GC2: CallInvalidEv GC1: CallCtlConnEstablishedEv P GC1: ConnDisconnectedEv B CiscoTransferEndEv

Scenario 25

Hunt list feature is disabled

Consult – Transfer Scenario

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A, B and C.	UnSupported Configuration

Hunt List Connected Number

Hunt pilot B configured with "Display Line Group Member DN as Connected Party" enabled. B has HL1 as its hunt list which has C and D as its hunt members

Scenario	Expected results	Call info
A calls B, Application is observing A only. GC1 is the GCID of the call. Call is offered on the huntmember C and C answers	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B	Call.getCurrentCallingAddress = A Call.getModifiedCallingAddress = A Call.getCurrentCalledAddress = null Call.getModifiedCalledAddress = null Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C. Application is observing C. GC1 is the GCID of the call. C answers the call	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC: GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnActiveEv TermC CG1: CallCtlTermConnTalkingEv termC	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C Application is observing A and C. GC1 is the GCID of the call. C answers the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1:CallCtlTermConnTalkingEv TermC	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C and then to D Application is observing A, C and D. GC1 is the GCID of the call. Call moves to D and D answers	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1: ConnCreatedEv D GC1: ConnAlertingEv D GC1: ConnDisConnEv C GC1: CallCtlConnDiscConnEv C GC1:TermConnDroppedEv TermC GC1: CallCtlTermConnDroppedEv TermC GC1: ConnConnectedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnActiveEv TermD CG1: CallCtlTermConnTalkingEv termD	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C Application is observing A, C and D. GC1 is the GCID of the call. C answers the call. C consults D and completes transfer	GC1: CallActiveEv GC1: ConnCreatedEv A ... GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1:CallCtlTermConnTalkingEv TermC GC2: CallActiveEv GC2: ConnCreatedEv C ... GC2: CallCtlTermConnTalkingEv A	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
	GC2: ConnCreatedEv D GC2: ConnInProgressEv D GC2: CallCtlConnOfferedEv D ... GC2: ConnAlertingEv D GC2: CallCtlConnAlertingEv D GC2:TermConnCreatedEv TermD GC2: TermConnRingingEv TermD GC2:CallCtlTermConnRingingEvTermD GC2:CallCtlTermConnTalkingEv TermD CiscoTransferStartEv GC1 ConnCreatedEv D GC1: CallCtlConnEstablishedEv D GC1:CallCtlTermConnTalkingEv TermD ... ... GC1 ConnDroppedEv C GC1 ConnDroppedEv B ... GC2 CallInvalidEv CiscoTransferEndEv	Call.getCurrentCallingAddress = C Call.getModifiedCalledAddress = C Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
A calls D. D calls HP B, call is offered on C	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1:TermConnCreatedEv TermD GC1: TermConnRingingEv TermD GC1: CallCtlTermConnRingingEvTermD GC1 CallCtlConnEstablishedEv D GC1:CallCtlTermConnTalkingEv TermD GC2: CallActiveEv GC2: ConnCreatedEv D ... GC2: CallCtlTermConnTalkingEv D GC2: ConnCreatedEv C GC2: ConnInProgressEv C GC2: CallCtlConnOfferedEv C GC2: ConnCreatedEv D GC2: CiscoHuntConnCreatedEv B GC2: ConnConnectedEv D GC2: CallCtlConnEstablishedEv A GC2: ConnConnectedEv B GC2: CallCtlConnEstablishedEv B	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
C answers the call D completes transfer	GC2: CallCtlConnAlertingEv C GC2:TermConnCreatedEv TermC GC2: TermConnRingingEv TermC GC2: CallCtlTermConnRingingEvTermC: GC2 CallCtlConnEstablishedEv C GC2 CallCtlTermConnTalkingEv termC CiscoTransferStartEv GC1 ConnCreatedEv C GC1 CiscoHuntConnectionCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: CallCtlConnEstablishedEv C GC1:CallCtlTermConnTalkingEv TermD ... ... GC1 ConnDroppedEv D ... GC2 CallInvalidEv CiscoTransferEndEv	Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = C Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Intercom

Configuration: terminal T1 has intercom line A with TargetDN B, label Bob, Unicode label UBob. Terminal T2 has intercom line B. Application provider has both T1 and T2 in control list.

C, Carol, UCarol is in the same intercom group as A, and B.

D, David, UDavid is not in the same intercom group as A, B and C.

Action	Result	Call info
Application opens provider, after provider comes in service, application issues provider.getIntercomAddresses()	JTAPI returns A and B as array of CiscoIntercomAddress.	N.A

Action	Result	Call info
Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTarget Label() request at A.	JTAPI will return B as target DN and Bob and UBob as target label.	N.A
Application issues CiscoIntercomAddress.getDafaultIntercomTargetDN(), CiscoIntercomAddress.getDefaultIntercomTargetLabel() and CiscoIntercomAddress.getDefaultIntercomUnicodeTargetLabel() request at A.	JTAPI will return B as target DN and Bob and UBob as target label.	N.A
Application issues CiscoIntercomAddres.setIntercomTarget(C, Carol, UCarol) on intercom address A. After successful response, Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A
Application1 is observing CiscoIntercomAddress A and has AddressObserverAdded to it. Application2 sets intercom target, label to C, Carol, UCarol.	App1 : AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL	N.A
After above step Application1 issues CiscoIntercomAddres.setIntercomTarget(B, Bob, UBob) on intercom address A.	Exception will be thrown to application as another application instance has already set the target to C, Carol, UCarol.	N.A
Intercom target DN and label for intercom address A is set to default, now application issues CiscoIntercomAddres.setIntercomTarget(D, David, UDavid) on intercom address A.	Exception will be thrown as D, David, UDavid is not in the same intercom group.	N.A
Application has set intercom target DN and label to C, Carol, UCarol for intercom address A. Now CTI Manager goes out of service, JTAPI failover to another CTIManager node. After intercom address A come back in service, JTAPI will restore intercom target DN and label to C, Carol, UCarol respectively. Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A

Action	Result	Call info
Application has set intercom target DN and label to C, Carol for intercom address A. Now CTI Manager goes out of service, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN, label and UnicodeLabel to C, Carol, UCarol respectively, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI.	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A
Application is connected to a CTIManager node, Cisco Unified Communications Manager node goes down, intercom device failsover to another Cisco Unified Communications Manager node, after intercom address comes back in service. CTIManager should restore intercom target Dn and label. Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A
Application is connected to a CTIManager node, Cisco Unified Communications Manager node goes down, intercom device failsover to another Cisco Unified Communications Manager node, after intercom address comes back in service. CTIManager tries to restore intercom target Dn and label, however due to race condition some other application has already set the target Dn and Label, hence CTI is not able to restore the intercom target DN and label.	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A

Action	Result	Call info
Application is observing intercom addresses A and B. A has target set to B. User initiates intercom call. Intercom call is successful.	CallObserver at A and B: CallActiveEv GC1 Cause: CAUSE_NORMALConnCreatedEv A, Cause: CAUSE_NORMALConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMALTermConnCreatedEv A- T1 Cause: CAUSE_NORMAL TermConnActiveEv A- T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv A - T1 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL ConnCreatedEv B, Cause: CAUSE_NORMAL ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtl Cause = CAUSE_NORMAL CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
User at B presses talkback softkey to get connected to intercom initiator.	CallObserver at A and B: TermConnActiveEv B - T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CiscoRTPOutputStartedEv – T2CiscoRTPInputStartedEv – T1	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null

Action	Result	Call info
Intercom address A has target defined as B. Application initiates an intercom call by calling interface Address.ConnectIntercom() with dialeddigit as empty. Intercom call is successful.	CallObserver at A and B : CallActiveEv GC1 Cause: CAUSE_NORMAL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL TermConnCreatedEv T1 Cause: CAUSE_NORMAL TermConnActiveEv T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv T1 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL ConnCreatedEv B Cause: CAUSE_NORMAL C ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
Application initiate TerminalConnection.join() request on TerminalConnection of B to talkback. Request is successful.	CallObserver at A and B : TermConnActiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CiscoRTPOutputStartedEv – T2 CiscoRTPInputStartedEv – T1	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
Application tried to put the intercom call on hold at A by issuing TerminalConnection.hold()	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to accept intercom call at intercom target by issuing connection.accept() at connection of B.	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to reject intercom call at intercom target by issuing connection.reject() at connection of B.	Intercom call will be disconnected.	N.A

Action	Result	Call info
Application tried to redirect intercom call by issuing connection.redirect() at connection of A or B.	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to park call by issuing connection.park() at Connection of A or B.	PlatformException will be thrown, intercom call stay connected.	N.A
Terminal T1 has intercom address A which has intercom target set to B. Terminal T2 has intercom address B and another address C. C is in call with D, GC1. A initiates intercom call to B, intercom call is auto-answered at B	No event to GC1 call, it will stay in Connected State.	N.A
Application tries to set forward on intercom address A by issuing CiscoIntercomAddress. setForwarding ()	PlatformException will be thrown.	N.A
Application tries to setRingerStatus on intercom address A by issuing CiscoIntercomAddress. setRingerStatus()	PlatformException will be thrown.	N.A
Application tries to setMessageWaiting on intercom address A by issuing CiscoIntercomAddress.setMessageWaiting()	PlatformException will be thrown.	N.A
Application tries to setAutoAcceptEnabled on intercom address A at CTIPort by issuing CiscoIntercomAddress. setAutoAcceptStatus ()	PlatformException will be thrown.	N.A
Application tries to getAutoAcceptEnabled on intercom address A at CTIPort by issuing CiscoIntercomAddress. getAutoAcceptStatus ()	PlatformException will be thrown.	N.A

DeviceState Whisper Scenario

Configuration: Terminal T1 has intercom address B, Terminal T2 has intercom address A. Application has set CiscoTermEvFilter to enable CiscoTermDeviceStateWhisperEv as well as all other DeviceState filters on T1 and T2. Application had added Terminal observer on both T1 and T2.

Action	Events	Call info
Intercom address A has target defined as B. Application initiates an intercom call by calling interface Address.ConnectIntercom() with dialeddigit as empty.	Event received at TerminalObserver of T1 CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL Event received at TerminalObserver of T2 CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_NORMAL	N.A

Action	Events	Call info
Application issue join() request on TerminalConnection of T2 (intercomTarget) to talkback to T1(intecomInitiator)	Event received at TerminalObserver of T1 None. Event received at TerminalObserver of T2 CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL	N.A
Terminal T2 already have intercom target call, Application enables CiscoTermFilter for CiscoTermDeviceStateWhisperEv.	**Event received at TerminalObserver of T2** CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_SNAPSHOT	N.A

iSac Codec

CiscoMediaTerminal Static Registration with iSac Codec

Actions	Events	Call info
1. Observe both A(CiscoMediaTerminal) and B Static Register A with media capability as CiscoMediaCapability. ISAC A calls B	CiscoTermInServirceEv for TA CiscoAddrInServiceEv for A	

Actions	Events	Call info
B answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoRTPOutputStartedEv for TB TA: CiscoRTPInputStartedEv for TA TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	(CiscoRTPInputStartedEv for TA).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TA).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TA).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getPacketSize() is not deterministic

CiscoMediaTerminal Dynamic Registration with iSac Codec

Actions	Events	Call info
1. Observe both A and B (CiscoMediaTerminal) Dynamic Register B with media capability as CiscoMediaCapability. ISAC	CiscoTermInServirceEv for TB CiscoAddrInServiceEv for B	

Actions	Events	Call info
A calls B	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB	

Actions	Events	Call info
B answers App sets RTP params on B	GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoMediaOpenLogicalChannelEv for TB TB: CiscoRTPOutputStartedEv for TA TA: CiscoRTPInputStartedEv for TB TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	CiscoMediaOpenLogicalChannelEv.getPayloadType() will return CiscoRTPPayload.ISAC CiscoMediaOpenLogicalChannelEv.getPacketSize() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPacketSize() is not deterministic

CiscoRouteTerminal Dynamic Registration with iSac Codec

Actions	Events	Call info
1. Observe both A and B (CiscoRouteTerminal) Dynamic Register B with media capability as CiscoMediaCapability. ISAC	CiscoTermInServirceEv for TB CiscoAddrInServiceEv for B	

Actions	Events	Call info
A calls B	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB	
B answers	GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoMediaOpenLogicalChannelEv for TB	CiscoMediaOpenLogicalChannelEv.getPayloadType() will return CiscoRTPPayload.ISAC CiscoMediaOpenLogicalChannelEv.getPacketSize() is not deterministic

Actions	Events	Call info
App sets RTP params on B	TB: CiscoRTPOutputStartedEv for TA TA: CiscoRTPInputStartedEv for TB TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	(CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPacketSize() is not deterministic

JTAPI Cisco Unified IP 7931G Phone Interaction

A and C are JTAPI application controllable Addresses. B1 and B2 are Address on Cisco Unified IP 7931G Terminal. Cisco Unified IP 7931G Terminal is configured to do Transfer across Addresses. B1 and B2 has shared Line B1’ and B2’ respectively configured on JTAPI controllable Terminal.

Action	Events	Call info
Scenario:1 Application is observing A: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer.	JTAPI Event received to CallObserver at A GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = null) GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1
Scenario:2 Application is observing A, B1': A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer	JTAPI Event received to CallObservers at A and B1' GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, TransferredCall = null) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
Scenario:3 Application is observing A, B1', B2': A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A and B1’ GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1’, FinalCall = GC1, TransferredCall = GC2) GC1- TermConnDroppedEv for TB1’ Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1’ Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for TB2’ Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2’ Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC-1 CiscoTransferEndEv	

Action	Events	Call info
Scenario:4 Application is observing A, B1', B2' and C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A, B1', B2' and C GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, TransferredCall = GC2) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause:	

Action	Events	Call info
	CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv	

Action	Events	Call info
Scenario:5 Application is observing C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = B2 Called = C CurrCalling = A CurrCalled = C LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at C GC1- CallActiveEv for callID = 101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoTransferStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER	

Action	Events	Call info
Scenario:6 Application is observing both A and C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = C LRP = B1

Action	Events	Call info
	JTAPI events at observer of A & C: GC-1CiscoTransferStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER NEW META GC-1 ConnDisconnectedEv for B1 –GC1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 – GC1 Cause: CAUSE_UNKNOWN CallControlCause:	

Action	Events	Call info
	CAUSE_TRANSFER	
Scenario:7 Application is observing A: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1
Scenario:8 Application is observing A, B1': A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv TB1' GC1 CallCtlTermConnBridgedEv TB1' GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
Scenario:9 Application is observing A, B1', B2': A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A, B1' and B2' GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv – TB1' GC1 CallCtlTermConnBridgedEv – TB1' GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
Scenario:10 Application is observing A, B1', B2', and C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference		Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A, B1', B2' and C GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv - TB1' GC1 CallCtlTermConnBridgedEv - TB1' GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for TC Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TC Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC-1 CiscoConferenceEndEv	

Action	Events	Call info
Scenario:11 Application is observing C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference.		Calling = B2 Called = C CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at C GC1- CallActiveEv for callID = 101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoConferenceStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for B1 Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B1 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B1 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	

Action	Events	Call info
Scenario:12 Application is observing both A and C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference.	JTAPI events at observer of A & C: GC-1CiscoConferenceStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRAN CAUSE_CONFERENCE SFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Locale Infrastructure Development Scenarios

Scenario 1—JTAPI Client Machine Has Connectivity to CallManager TFTP Server

- During install, JTAPI client would prompt user to enter TFTP IP address.
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection.
- User can select language for running JTAPI Preference application.
- JTAPI Preference application is run second time, it will present UI in the language that user selected before.

Scenario 2—JTAPI Client Machine Doesn't Have Connectivity to CallManager TFTP Server

- During install JTAPI Client would prompt user to Enter TFTP-IP Address
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection but user will have only English language to select.
- JTAPI Preference application is run second time, it will present UI in the English languages.
- TFTP connectivity is restored. Now JTAPI Preferences UI is run, it will take user to language selection

Scenario 3—JTAPI Client Machine Has Connectivity to CallManager TFTP Server

- During install JTAPI Client would prompt user to Enter TFTP-IP Address
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection.
- User can select language for running JTAPI Preference application.
- JTAPI Preference application is run second time, it will present UI in the language that user selected before.
- Now new locale files are available with added support for a new languages.
- User runs JTAPI Preferences application, JTAPI Preferences application would notify user about available.
- Application restart JTAPI Preferences application, user will be support for new language.

Calling Party Normalization

Scenario 1—Incoming Call From a PSTN Number (Local) to JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a PSTN Number [55555555] A & the Number type is [Subscriber] through the gateway to a JTAPI Observed Terminal [2222] B.	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (55555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Subscriber

Scenario Two—Incoming Call From a National PSTN Number to JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a Dallas PSTN Number [55555555] A & the Number type is [National] through a gateway to a JTAPI Observed Terminal [2222] B.	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (97255555555) Called: B (2222) getModifiedCallingAddress (): 97255555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +197255555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: National

Scenario Three—Incoming Call From Inter-National PSTN Number to JTAPI Observed Terminal

Action	Events	Call info
A Call is offered from India PSTN Number [918028520261] & the Number type is [Inter-national] through a San Jose Gateway to a JTAPI observed Terminal [2222]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (918028520261) Called: B (2222) getModifiedCallingAddress (): 918028520261 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +918028520261 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Inter-National

Scenario Four—Outgoing Call From JTAPI Observed Terminal to PSTN Number [SUBSCRIBER]

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to a PSTN number [44444444] and the Number type is [SUBSCRIBER]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (44444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 44444444 getCurrentCalledAddress(): 44444444 getCurrentCalledPartyInfo(): 44444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType (). getNumberType () would return: Unknown.

Scenario Five—Outgoing Call From JTAPI Observed Terminal to National PSTN Number

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to a Dallas PSTN number [97244444444] & the Number type is [NATIONAL]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (97244444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 97244444444 getCurrentCalledAddress(): 97244444444 getCurrentCalledPartyInfo(): 97244444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknown.

Scenario Six—Outgoing Call From JTAPI Observed Terminal to International PSTN Number

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to India PSTN number [918028520261] & the Number type is [INTERNATIONAL]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (918028520261) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 918028520261 getCurrentCalledAddress():918028520261 getCurrentCalledPartyInfo(): 918028520261 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknown.

Scenario Seven—Incoming Call From PSTN Redirected to Another PSTN by JTAPI Observed Terminal

Action	Events	Call info
A call is offered from PSTN [55555555] through a San Jose	NEW META EVENT_____META_CALL_STARTING	Calling: A (55555555)Called: B (2222)

Action	Events	Call info
Gateway to a JTAPI observed terminal [2222] which redirects the call to another San Jose PSTN [44444444]. In CallState [Idle] the fwdDestination Address (Redirect Address) should be a minus (-).	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause:CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL CallRedirectReq Redirect Address = C CallRedirectRes ConnCreatedEv at C Cause: CAUSE_REDIRECTED ConnInProgress Calling party:A, Called Party: C, LRP: B CallRedirectRes CallStateChangedEv (IDLE) Reason: REDIRECT	getModifiedCallingAddress (): 55555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: SUBSCRIBER destinationAddress: 44444444. getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknowns

Scenario Eight—Incoming Call From PSTN Number (Local) to JTAPI Observed Terminal Who Transfers to Another JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a PSTN Number [55555555] A & the Number type is [Subscriber] through a San Jose gateway to a JTAPI observed Terminal [1111] X which transfers the call to another JTAPI Observed Terminal [2222] B	After Transfer: GC1: CiscoTransferStartEv ConnCreatedEv for B ConnConnectedEv for B CallCtlConnEstablishedEv for B TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv GC2: CiscoTransferStartEv TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv	After Transfer: Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (+140855555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Subscriber

Click to Conference

A, B, C and D are addresses and TermA, TermB, TermC and TermD are corresponding terminals.

Action	Events	Call info
A and B are in a call GC1 created using click-to-call. User adds C to the conference call. GC2 is the initial call at C. Application is observing only C	GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: ConnInProgressEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	callingAddress = unknown calledAddress = C CurrentCalling = unknown CurrentCalled = C
	GC2: CallCtlConnOfferedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	

Action	Events	Call info

Action	Events	Call info
	GC2: ConnAlertingEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlConnAlertingEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnCreatedEv TermC GC2: TermConnRingingEv TermC CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnRingingEv TermC CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermC GC1: TermConnRingingEv TermC CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: ConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC2: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallInvalidEv CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv C CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL GC1: TermConnTalkingEv TermC CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL	

Action	Events	Call info
A calls B using click-to-call – GC1. A adds C to the call using click-2-conf. Application has call observer on A		Calling address: A Called address: B Current calling: A Current called: B Last redirecting party = null After C is conferenced, callinfo is not applicable.

Action	Events	Call info
	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER GC1: ConnCreatedEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnInProgressEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermA GC1: TermConnRingingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv A CiscoFeatureReason = REASON_NORMAL cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL TermConnActiveEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnTalkingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnInProgressEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv B CiscoFeatureReason = REASON_NORMAL: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	
A consults with D-GC3. A completes conference. The events received by application remains the same (as that of consult conference).	GC1: TermConnHeldEv TermA GC3: ConsultCallActiveEv GC3: ConnCreatedEv A GC3: ConnCreatedEv D GC3: CallCtlConnAlerting D GC3: ConnConnectedEv D GC3: CallCtlConnEstablishedEv B CiscoConferenceStartEv GC3->GC1 GC3: CallCtlConnDisconnectedEv A GC3: CallCtlConnDisconnectedEv D GC1: ConnCreatedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnTalkingEv TermA GC3: CallInvalidEv CiscoConferenceEndEvent	For consult call GC3: Calling address: A Called address: D Callinfo not applicable after conference is completed.
User drops D using click-2-conf feature User drops C using click-2-conference interface	GC1: ConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	Calling address: A Called address: B

Action	Events	Call info
Drop all parties a conference. A calls B using click-2-call. User adds C to the conference using click-2-conference. All parties are dropped using click to conference. Application has call observers on A, B and C.	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER GC1: ConnCreatedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	Calling address: A Called address: B GC2: Calling address = unknown Called address: C

Action	Events	Call info

Action	Events	Call info
	GC1: ConnInProgressEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnOfferedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: ConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: TermConnCreatedEv TermA GC1: TermConnRingingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnRingingEv TermA GC1: ConnConnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnActiveEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: ConnCreatedEv B Cause: CAUSE_NORMAL CiscoFeatureReason:REASON_REFER GC1: ConnInProgressEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: ConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnCreatedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

Action	Events	Call info
	GC1: TermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnActiveEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

Action	Events	Call info
	GC2: CiscoCallChangedEv GC2->GC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: ConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallInvalidEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	

Action	Events	Call info
	GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnActiveEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnTalkingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE All parties are dropped using click-2-conference GC1: TermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: TermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE	

Action	Events	Call info
	GC1: CallCtlTermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallInvalidEv	
A calls B using click-to-call – GC1. A adds C to the call using click-2-conf. User drops party C. Application has call observer on C only.	GC1: TermConnDroppedEv TermC CiscoFeatureReason = REASON_CONFERENCE	NA
	GC1: CallCtlTermConnDroppedEv TermC CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv A CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv A CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv B CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv B CiscoFeatureReason = REASON_CONFERENCE GC1: CallInvalidEv	
A calls B using GC1. Address C is configured on TermC1 and TermC2. Application has call observer on C	GC2: CallActiveEv CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE	Calling = unknown Called = C Last redirecting = null
User uses click-to-conference to C.	GC2: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE	

Action	Events	Call info

Action	Events	Call info
	GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE CiscoCallChangedEv GC2->GC1 TermConn TermC1 CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason:	

Action	Events	Call info
	REASON_CLICK_TO_CONFERENCE CiscoCallChangedEv GC2->GC1 TermConn TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallInvalidEv GC1: ConnCreatedEv B Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnConnectedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	

Action	Events	Call info
	GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnPassEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnInUseEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	C answers at TermC1

Call Pickup

The basic test case for the fix was the following:

1. B and C are devices in a call pick up group. A is a device not in it.
2. A calls B.
3. C goes off-hook, and presses the Pickup softkey.
4. C is now on the call with A.

This test was run with variations in which devices were observed, and the full matrix was run. This included:

- Observing A, B, and C
- Observing A and B
- Observing A and C
- Observing B and C
- Observing only A
- Observing only B
- Observing only C

The final test run, observing only C, was the primary concern for this fix, based on customer usage. The feature request was, when only observing C, being able to get information about the original called party (A) on Pickup. All test cases passed and the correct information was displayed for all of them.

For cases 3 and 4, the call information depends on the order of events that JTAPI delivers. If JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events before the
GC2-CiscoCallChangedEv, then the call information, such as calling and called addresses, will be what was seen in
GC2. Conversely, if JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events after the
GC2-CiscoCallChangedEv, then the call information, such as calling and called addresses, will be what was seen in
GC1.

As an example, if JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events before the
GC2-CiscoCallChangedEv, the Calling Address = C, Called Address = Pickup Number. If JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events after the
GC2-CiscoCallChangedEv, the Calling Address = A, Called Address = B.

These test cases were run with auto-pickup enabled and disabled, and there was much difference in the functionality of the two. Most of the test cases are enumerated below.

The basic call from A to B is the same in all cases, and is only shown in the first case below.

Scenario One

Observing all devices and auto-pickup enabled.

Action	Events	Call info (GCID info)
A goes off-hook and dials B (Basic Call) B is ringing. C goes off-hook and presses Pickup softkey. Connection for C is dropped, B is dropped / cleaned up, C connection on Call 1 is established		Calling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B CCalling: C, CCalled: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE REASON_CALLPICKUP CCalling: A, CCalled: C LRP: NONE REASON_CALLPICKUP CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL REASON_CALLPICKUP CCalling: A, CCalled: C REASON_NORMAL

Action	Events	Call info (GCID info)
	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv	

Action	Events	Call info (GCID info)
	GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallCtlConnEstablishedEv-C	

Note Both B and C in the following scenarios have exactly the same behavior and events. Only the behavior of device C (the one picking up the call) changes.

Scenario Two

Observing all devices with auto-pickup disabled.

Action	Events	Call ID Info
C goes off-hook and presses Pickup softkey	GC2-CallActiveEvent	CCalling C, CCalled: NONE
Call 2 gets dropped or invalidated	GC2-ConnCreatedEvent-C	LRP: NONE
C gets a connection on Call 1	GC2-ConnConnectedEvent-C	REASON_NORMAL
B is dropped from Call 1	GC2-CallCtlConnInitiatedEv-C	REASON_CALLPICKUP
C is ringing	GC2-TermConnCreatedEvent	CCalling A, CCalled: C
C is on call with A	GC2-TermConnActiveEvent	Calling: A, Called: C, LRP: B
	GC2-CallCtlTermConnTalkingEv	REASON_CALLPICKUP
	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	Calling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP REASON_NORMAL REASON_NORMAL
	GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C	
	GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	
	GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	
	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	

The flow of events differs greatly when the auto-pickup option is enabled or disabled. When Auto Call Pickup is disabled and a user presses the Pickup softkey (C), the phone rings. The user has to answer the phone as if it is a normal call. When the phone is ringing, the original call that was created when they went offhook is terminated, they are connected to the existing call, and the old party (B) is removed from the call. There is no CiscoCallChangedEv generated when Auto Call Pickup is disabled, because the call does not change, it is terminated before C joins the new call.

A Group Pickup scenario follows, during which the Group Pickup softkey is used in place of the Pickup softkey. This required actually dialing the number for the pickup group. Group Pickup also is subject to the Auto Call Pickup service parameter. The general flow and call events are identical to the normal Call Pickup scenarios, except with added events for the required dialing of the pickup number.

Scenario Three

Observing all devices with group pickup and auto-pickup enabled.

Action	Call event	Call ID Info
C goes offhook and presses Group Pickup softkey C is dialing the PU Number C is added to the original call Pickup added to original call Pickup # is removed Call 2C is dropped from Call 2Pickup # is removed Call 1 B is dropped / invalidated	GC1 [add to others to clarify] GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv **GC2-CallCtlConnDialingEv-C** **GC2-ConnCreatedEvent-PU** **GC2-ConnInprogressEvent-PU** **GC2-CallCtlConnEstablishedEv-C** GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C **GC1-ConnCreatedEvent-PU** GC1-ConnConnectedEvent-C **GC1-CallCtlConnEstablishedEv-C** GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv **GC1-ConnInprogressEvent-PU** **GC1-CallCtlConnOfferedEv-PU** **GC2-ConnDisconnectedEvent-PU** **GC2-CallCtlConnDisconnectedEv-PU** GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP:B REASON_CALLPICKUP, LRP: PU CCalling: C, CCalled: PU REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP

There are only a handful of changes for the above Group Pickup case, and they all directly relate to the extra required step of dialing the pickup number.

Scenario Four

Observing all devices with Group Pickup and Auto-Pickup disabled.

Action	Event	Call info
C goes offhook and pressed “Group Pickup” softkey C is dialing the PU number PU is removed from Call 2 C is removed from Call 2 Call 2 is destroyed C gets a connection on Call 1 B is dropped from Call 1 C is ringingC picks up	GC1 GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent[ADDRS] GC1-ConnInprogressEvent GC1-CallCtlConnOfferedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent GC1-CallCtlConnDisconnectedEv GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

The tables above have scenarios during which all of the devices were observed. The devices were run with every possible combination, across all varieties of Pickup and Group Pickup. Parts of the scenarios had the exact same output and others were redundant and are not shown here. For example, device A and B were identical and shown only once.

Scenario Five

Only observing device B.

Action	Call events	Call IDs/Call info
A is in the process of calling B B is ringing A is removed from Call 1 B is removed from Call 1	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallInvalidEvent GC1-CallObservationEndedEv	CCalling: A, CCalled: B, Calling: A, Called: B, LRP: NONE REASON_NORMAL REASON_CALLPICKUP REASON_NORMAL

Scenario Six

Observing only device A.

Action	Call events	Call IDs/Call info
A goes offhook and dials B B is ringing C is ringing B is removed from Call 1	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C	CCalling: A, CCalled: NO, NO LRP REASON_NORMAL CCalling A, CCalled B, Called: NOT SET LRP: NONE CCalling: A, CCalled: C, LRP: B Called: NOT SET REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_NORMAL

Scenario Seven

Observing only device C with Auto-Pickup enabled.

Action	Call events	Call IDs/Call info
C goes offhook and presses “Pickup” hotkey C is connected to Call 1 C is dropped from Call 2Call 2 is invalidated / cleared A and C are connected on Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-CallCtlConnEstablishedEv-C	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REAON_CALLPICKUP CCalling A, CCalled: NONE LRP: NONE CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_CALLPICKUP REASON_CALLPICKUP CCalling A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

Scenario Eight

Observing only device C with Auto-Pickup disabled.

Action	Call events	Call IDs/Call info
C goes offhook and pressed “Pickup” softkey Call 2 is destroyed C is added to Call 1, but does not pick upC is ringing C picks up, and is connected to Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LR PREASON_NORMAL REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Scenario Nine

Observing only device C with Group Pickup and AutoPickup enabled.

Action	Call event	Call IDs/Call info
C goes offhook and presses “Pickup” softkey C dials the Pickup Number C is added to Call 1PU is added to Call 1 PU # is removed from Call 2 C is removed from Call 2Call 2 I invalidated / cleared C is connected to Call 1 PU is removed from Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-PU GC1-CallCtlConnEstablishedEv-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PUCCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP CCalling: A, C Called: C CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKU PREASON_NORMAL CCalling: A, CCalled: C REASON_CALLPICKUP CCalling: A, CCalled: C REASON_CALLPICKUP

Scenario Ten

Observing only device C with Group Pickup and Auto-Pickup disabled.

Action	Call events	Call IDs/Call info
C goes offhook, and presses “Group Pickup” softkey C dials the PU Number PU is dropped from Call 2C is dropped from Call 2Call 2 is destroyed C is added to Call 1 C is ringing C is connected to A	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC1-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_CALLPICKUP REASON_NOTMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

selectRoute() with Calling Search Space and Feature Priority

The selectRoute() API with calling search space and feature priority as array of int. is shown in the following table.

Action	Events	Call info
Add call observer on phones A, B, C, D. Register Route Point RP Register route call back, with select route API with three rows route selected: A, ...CSS: 0, FP: 1 route selected: B, ...CSS: 1, FP:3 route selected: D, ...CSS: 1, FP:1 C calls RP Call rings at A. A answers. C-A call is connected.	GC1 CallActiveEv GC1 ConnCreatedEv C: GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C: GC1 TermConnCreatedEv TC GC1 TermConnActiveEv TC GC1 CallCtlTermConnTalkingEv TC GC1 CallCtlConnDialingEv C: GC1 CallCtlConnEstablishedEv C: GC1 ConnCreatedEv RP: GC1 ConnInProgressEv RP: GC1 CallCtlConnOfferedEv RP: After redirect request is processed GC1 ConnCreatedEv A: GC1 ConnInProgressEv A: GC1 CallCtlConnOfferedEv A: GC1 ConnDisconnectedEv RP: GC1 CallCtlConnDisconnectedEv RP: GC1 ConnAlertingEv A: GC1 CallCtlConnAlertingEv A: GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnConnectedEv A: GC1 CallCtlConnEstablishedEv A: GC1 TermConnActiveEv A GC1 TermConnActiveEv A [C] CiscoRTPInputStartedEv [A] CiscoRTPOutputStartedEv [A] CiscoRTPInputStartedEv [C] CiscoRTPOutputStartedEv	calling: C lastRedirected:RP called: A

Extension Mobility Login Username

Terminal A is in control list of user, Terminal B is not in control list of User. Extension Mobility login username is John, end user id user for application is John.

Action	Result	Call info
Open provider, Terminal A doesn't have any observer, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A.	InvalidStateException is thrown.	NA
Open provider, Add Observer to Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A.	Application should get empty string “” for username.	NA
Open provider, User “John” EMLogin to Terminal A and add observer to the Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal A, now open provider, add observer to Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal A, now open provider, add observer to Terminal A, User “John” EMLogout of Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get empty string “” for username	NA

Action	Result	Call info
OpenProvider, User “John” EMLogin to Terminal B, add observer, application calls CiscoTerminal.getEMLoginUserName() at Terminal B **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal B, OpenProvider, add observer to Terminal B, Application calls CiscoTerminal.getEMLoginUserName() at Terminal B **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA

Terminal A is in control list of user and configured with the Extension Mobility logout profile of user Kerry. The Kerry profile is configured with logout username as Kerry. There is another profile with login username of John.

Action	Result	Call info
User John logs into Terminal A, OpenProvider and add observer to Terminal A. Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. John logs out at Terminal A. Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String John. Application should get String Kerry.	NA NA

Calling Party IP Address

The following are some examples of call scenarios.

Basic Call Scenario

- JTAPI application monitors party B
- Party A is an IP phone
- A calls B

- IP Address of A is available to JTAPI application monitoring party B

Consultation Transfer Scenario

- JTAPI application monitors party C
- Party B is an IP phone
- A talking B
- B initiates a consultation transfer call to C
- IP Address of B is available to JTAPI application monitoring party C.

Consultation Conference Scenario

- JTAPI application monitors party C
- Party B is an IP phone
- A talking B
- B initiates a consultation conference call to C
- IP Address of B is available to JTAPI application monitoring party C.

Redirect Scenario

- JTAPI application monitors party B and party C
- Party A is an IP phone
- A calls B
- IP Address of A is available to JTAPI application monitoring party B
- Party A redirects B to party C (
- Calling IP address is not available to JTAPI application monitoring party B (not a supported scenario).
- Calling IP address of B is provided to JTAPI application monitoring party C.

CiscoJtapiProperties

1. Set Socket Connect Timeout to 5 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager and do a normal provider open. Expected Result: Socket Connect to Primary CTI Manager should fail in not more than 5 secs
2. Set Socket Connect Timeout to 5 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager, set security options to True and do a secured provider open. Expected Result: Socket Connect to Primary CTI Manager should fail in not more than 5 secs (Socket Connect timed-out in ~5 seconds, though it took some additional time initially for verifying security certificates)
3. Set Socket Connect Timeout to 0 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager, set security options to true and do a secured provider open. Expected Result: Socket Connect to Primary CTI

Manager will no longer rely on new Service Parameter (Socket Connect timed-out in ~23 seconds, though it took some additional time initially for verifying security certificates).

IPv6 Support

Use Case1 - Basic Call Scenario: Calling Is IPv6 Enabled Phone; Called Is IPv6

Action	Events	Call info/Expected result
IPv6 enabled phone A calls JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoOutputStartedEv will contain the far-end Ipv6 RTP address(of A) getRemoteAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone(B)

Action	Events	Call info/Expected result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case2 - Basic Call Scenario: Calling Is IPv6 Enabled Phone; Called Is IPv4

Action	Events	Call info/Expected result
IPv6 enabled phone A calls JTAPI Observed IPv4 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv4 RTP address of the monitored phone

Action	Events	Call info/Expected result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case3 - Basic Call Scenario: Calling Is IPv4 Enabled Phone; Called Is IPv6

Action	Events	Call info/Expected Result
IPv4 enabled phone A calls JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object. CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case4 - Basic Call Scenario: Calling Is IPv4 Enabled Phone; Called Is IPv4

Action	Events	Call info/Expected Result
IPv4 enabled phone A calls JTAPI Observed IPv4 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object. CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv4 RTP address. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv4 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case5 - Consultation Transfer Scenario, IPv6 Device Consults

Action	Events	Call info/Expected Result
GC1: Call between A & B Consult Call: IPv6 enabled phone B consults JTAPI Observed device C for Transfer using GC2.	NEW META EVENT_____META_CALL_STARTING	For Consult Call: CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for B in an InetAddress object to the JTAPI Application observing C. While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return null
C Answers	**CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case6 - Consultation Transfer Scenario, IPv4 Device Consults

Action	Events	Call info/Expected Result
GC1: Call between A & B Consult Call: IPv4 enabled phone B consults JTAPI Observed device C for Transfer using GC2.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for B in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null
C Answers	**CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case7: Redirect Scenario

Action	Events	Call info/Expected Result
GC1: Call between A(IPv6) & B Redirect Call: phone B redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return null

Action	Events	Call info/Expected Result
C Answers		

Action	Events	Call info/Expected Result
	CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause :	

Action	Events	Call info/Expected Result
	CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case8: Redirect Scenario (IPv4)

Action	Events	Call info/Expected results
GC1: Call between A(IPv4) & B Redirect Call: phone B redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null

Action	Events	Call info/Expected results
C Answers		

Action	Events	Call info/Expected results
	CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause :	

Action	Events	Call info/Expected results
	CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case9: Redirect Scenario, Calling Device Redirects

Action	Events	Call info/Expected Result
GC1: A calls B(IPv4) Redirect Call: Phone A redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING **CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for B in an InetAddress object to the JTAPI Application observing C CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() or, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr()_v6 **will not return** IP address for A in an InetAddress object to the JTAPI Application observing B after redirect.

Action	Events	Call info/Expected Result
C Answers		

Action	Events	Call info/Expected Result
	ConnCreatedEv for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for A Cause : CAUSE_NORMAL **ConnDisconnectedEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for A Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL	

Action	Events	Call info/Expected Result
	TermConnActiveEv for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case10: Route Scenario, IPv6 Enabled Calls RoutePoint Which Routes Call to IPv6 Device

Action	Events	Call info/Expected Result
IPv6 enabled phone A calls RoutePoint which routes the call to JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoRouteEvent.getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. While, CiscoRouteEvent. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv6 RTP address getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallRouteEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case11: Enterprise Parameter "Enable IPv6" is Enabled

Application does an open provider by providing the list of CTI Manager IPs as

- IPv4 address of CTI Manager1
- IPv6 address of CTI Manager1
- IPv4 address of CTI Manager2
- IPv6 address of CTI Manager2

Now once the JTAPI is able to establish a connection with CTI Manager and later on if CTI Manager1 goes down, in failover attempt application can see delay in connecting as JTAPI will first try to connect with IPv6

address of CTI Manager1 (which is next in the list) even though that IP address is of the same CTI Manager and only once it times out it will try with the IPv4 address of the CTI Manager2 which will succeed (assuming CTI Manager2 is running).

Provider Open Scenario

1. Service Parameter for Reconnect Attempt is not set(or set to 0), Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is stared again and connection is restored.
   - Enterprise parameter “Enable IPv6”is enabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address.
2. Service Parameter for Reconnect Attempt is not set (or set to 0), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is stared again and connection is restored.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address. But, the existing devices registered with IPv6 address will be closed with “CiscoTermRegistrationFailedEv” with a new reason code “IP_CAPABILITY_MISMATCH”
3. Service Parameter for Reconnect Attempt is not set (or set to 0), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv6 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is started again and connection is restored.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will not be able to reconnect to CTI Manager, as it no longer supports IPv6 address but JTAPI will try reconnecting to CTI Manager indefinitely till the time service parameter is again enabled and CTI Service restarted.
4. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
   - Enterprise parameter “Enable IPv6”is enabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address.
5. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI Manager.

- CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
- Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address. But, the existing devices registered with IPv6 address will be closed with “CiscoTermRegistrationFailedEv” with a new reason code “IP_CAPABILITY_MISMATCH”

6. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv6 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will not be able to reconnect to CTI Manager, as it no longer supports IPv6 address but JTAPI will try reconnecting to CTI Manager 5 more times (as the same can again be enabled on Cisco Unified Communications Manager) before closing all the devices and provider.
7. Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv6 address. JTAPI will not be able to open a connection with CTI manager. Retry attempts are applicable only if connection gets established once, but since in this scenario even the first attempt is failing so there will be no subsequent reconnect attempts.

   Enterprise parameter “Enable IPv6” is enabled. Application does an open provider by providing the list of CTI Manager IPs as
   - IPv4 address of CTI Manager1
   - IPv6 address of CTI Manager1
   - IPv4 address of CTI Manager2
   - IPv6 address of CTI Manager2

   Now once the JTAPI is able to establish a connection with CTI Manager and later on if CTI Manager1 goes down, in failover attempt application can see delay in connecting as JTAPI will first try to connect with IPv6 address of CTI Manager1 (which is next in the list) even though that IP address is of the same CTI Manager and only onMangerce it times out it will try with the IPv4 address of the CTI Manager2 which will succeed (assuming CTI Manager2 is running).

Calling Party IP Address Scenarios

1. Ipv6 enabled phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv6 calling party IP address.

   getCallingPartyIpAddr() will return NULL

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address.
2. Ipv4 enabled phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 calling party IP address (existing behavior)

getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

getCallingPartyIpAddr_v6() will return NULL.

3. Ipv6 only phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv6 calling party IP address.

   getCallingPartyIpAddr() will return NULL

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

4. Ipv4 enabled phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 formatted calling party IP address.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return NULL.

5. Ipv4_v6(Two Stack) phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 and Ipv6 calling party IP addresses.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

6. Ipv4_v6(Two Stack) phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 and Ipv6 calling party IP addresses.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

RTP Addresses

1. An Ipv6 enabled phone calls an Ipv6 JTAPI Observed phone and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.

2. An Ipv4 enabled phone calls an Ipv4 JTAPI Observed phone and the call is answered. JTAPI will generate(existing behavior):
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.

3. An Ipv4 enabled phone calls an Ipv6 JTAPI Observed device and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.

4. An Ipv6 enabled phone calls an Ipv4 JTAPI Observed device and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
5. A Dual stack(Ipv4_v6) phone calls another dual stack(Ipv4_v6) JTAPI Observed device, preferred media termination is set to IPv6, and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
6. A Dual stack(Ipv4_v6) phone calls another dual stack(Ipv4_v6) JTAPI Observed device, preferred media termination is set to IPv4, and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
7. A Dual stack(Ipv4_v6) phone calls an Ipv4 JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
8. A Dual stack(Ipv4_v6) phone calls an Ipv6 JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
9. An IPv4 phone calls a dual stack (Ipv4_v6) JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
10. An IPv6 phone calls a dual stack (Ipv4_v6) JTAPI Observed device and the call is answered then JTAPI will generate:
    - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
    - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
11. JTAPI observed IPv6 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for Transfer. C answers and B completes the Transfer, then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of C.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of A.

- At C:
  - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of A.
  - CiscoRTPInputStartedEv containing the Ipv4 RTP address of C.

12. JTAPI observed IPv4 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for Transfer. C answers and B completes the Transfer, then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv4 RTP address of A.
    - At C:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of C.
13. JTAPI observed IPv6 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for conference. C answers and B completes the conference. Conference Bridge has an IPv4 address. Then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of A.
    - At B:
      - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the Conference Bridge.
      - CiscoRTPInputStartedEv containing the Ipv4 RTP address of B.
    - At C:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of C.

CTI Port/Route Point Registration Scenarios

1. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv6 address and Application addressing capability as IPv6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv6 address.

2. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 address and Application addressing capability as IPv4. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 address.
3. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 and IPv6 addresses and Application addressing capability as IPv4_v6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 and IPv6 addresses.
4. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 address and Application addressing capability as IPv4. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 address.
5. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv6 address and Application addressing capability as IPv6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv6 address.
6. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only’. Application tries a static register of that CTI Port/Route Point by providing an IPv6 address or/and by advertising application addressing capability as IPv6 (or Ipv4_v6) only then request will fail with a CiscoRegistrationException.
7. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4 (or IPv4_v6) only. Then the request will be denied with a CiscoRegistrationException.
8. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only (or IPv4_v6 both)’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4 only. Then the registration will succeed and CTI Port/Route point will get registered with IPv4 address when the same is provided with SetRTPParams request.
9. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only (or IPv4_v6 both)’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv6 only. Then the registration will succeed and CTI Port/Route point will get registered with IPv6 address when the same is provided with SetRTPParams request.
10. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6 both’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4_v6 both. Then the registration will succeed and CTI Port/Route point will get registered with IPv4_v6 address when the same is provided with SetRTPParams request.
11. If an application tries to dynamically register a CTI Port/Route Point by advertising its IP capabilities as IPv6, which is already registered to another application with IPv4 address. Then the request will be declined with a CiscoRegistrationException or “CiscoTermRegistrationFailedEv” will be sent with new reason code “IP_CAPABILITY_MISMATCH”.

Advance Test Cases

1. Application does a provider Open with IPv4 address to a CTI Manager which has enterprise parameter “Enable IPv6” enabled. Application tries to register a CTI Port/Route point with an IPv6 address whose device IP Addressing Mode is set to “IPv4_v6” by advertising applications addressing capability as “IPv6 only”. The registration request will succeed.
2. JTAPI observed IPv6 device A calls another JTAPI observed IPv4 device B, call is offered and answered at B. In that case:

   CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() will return NULL

   CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() will the actual calling Party IPv6 address

   - At B:
     - CiscoRTPInputStartedEv will have B’s IPv4 Address
     - CiscoRTPOutputStartedEv will have IPv4 address of the MTP Resource

   Interesting thing to notice here is CiscoRTPOutputStartedEv has an IPv4 address while calling party IP Address is an IPv6 address.

Direct Transfer Across Lines Use Cases

Action	Events	Call info/Expected Result
Application is observing A, B1, B2, and C (B1 and B2 are two Addresses on the same Terminal TB) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing A, B1, B2, and C (B1 and B2 are two Addresses on the same Terminal which allows connected transfer across lines over phone manually which supports this feature) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2		CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
User B2 presses transfer and user selects active call(A–> B call) from the phone UI and presses transfer again to do connected Transfer Across Lines	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B2 GC3: ConnConnectedEv for B2 GC3: CallCtlConnInitiatedEv for B2 GC3: TermConnCreatedEv for TB2 GC3: TermConnActiveEvent for TB2 GC3: CallCtlTermConnTalkingEv for TB2	

Action	Events	Call info/Expected Result
User Presses transfer again to complete connected transfer across lines	GC3: TermConnDroppedEv for TB2 GC3: CallCtlTermConnDroppedEv for TB2 GC3: ConnDisconnectedEv for B2 GC3: CallCtlConnDisconnectedEv for B2 GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Action	Events	Call info/Expected Result
Application is observing A, B1, B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
Application is observing B1, B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEv GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEvGC1: CallObservationEndedEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing only B1: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	JTAPI will throw PlatformException “Transfer controller is not set and could not find a suitable TerminalConnection”. Since JTAPI cannot get/find call leg for B2 from GC2	

Action	Events	Call info/Expected Result
Application is observing only A: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 User presses transfer and selects active call(A –> B call) from the phone UI and presses Transfer again to do Connected Transfer Across Lines	GC2: CallActiveEvent GC2: ConnCreatedEv for A GC2: ConnCreatedEv for C GC1: CiscoCallChangedEv GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for A GC2: TermConnActiveEvent for A GC2: CallCtlTermConnTalkingEv for A GC2: ConnConnectedEv for C GC2: CallCtlConnEstablishedEv for C GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: TermConnDroppedEv for A GC1: CallCtlTermConnDroppedEv for A GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: CallInvalidEvent GC1: CallObservationEndedEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing only B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	JTAPI will throw PlatformException “Transfer controller is not set and could not find a suitable TerminalConnection” Since JTAPI cannot get/find call leg for B1 from GC1	
Application is observing only C: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 User presses transfer and selects active call(A–>B call) from the phone UI and preses Transfer again to do Connected Transfer Across Lines.	GC2: CiscoTransferStartEv GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for AGC2: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
New user role(Standard Supports Connected Xfer/Conf) is associated with application user Application opens a provider and disassociates the above mentioned user role	JTAPI delivers: ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoTermRestrictedEv CiscoAddrRestrictedEv (for all the phones that support connected tx/conf across lines)	CiscoProviderCapabilityChangedEv. hasConnectedTransfer ConferenceCapabilityChanged() returns True

Action	Events	Call info/Expected Result
Application is observing A, B1, B2, C and C' (B1 and B2 are two Addresses on the same Terminal, C' is sharedline of C) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2 setTransferController to B1 GC1.transfer(GC2)	At Transfer: GC1: CiscoTransferStartEvGC2: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC' GC1: TermConnPassiveEvent for TC' GC1: CallCtlTermConnInUseEv for TC' GC2: TermConnDroppedEv for TC' GC2: CallCtlTermConnDroppedEv for TC' GC2: CiscoCallChangedEv GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
New user role(Standard Supports Connected Xfer/Conf) is not associated with application user Application tries to add observer on phone which allows connected transfer/conference across lines	Phones that allow connected transfer/conference across lines are exposed as restricted. JTAPI throws PlatformExceptionImpl ("Terminal is restricted", CiscoJtapiException.CTIERR_DEVICE_RESTRICTED).	CiscoTerminal.isRestricted() returns TRUE
New user role(Standard Supports Connected Xfer/Conf) is not associated with application user Application opens a provider and associates the above mentioned user role	JTAPI delivers: ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoAddrActivatedEv CiscoTermActivatedEv (for all the phones that support connected tx/conf across lines)	CiscoProviderCapabilityChangedEv. hasConnectedTransferConference CapabilityChanged() returns True

Connected Conference or Join Across Lines Use Cases - New Phones Behavior

Action	Events	Call info/Expected result
New Role “Standard Supports Connected Xfer/Conf” to control phones which support connected conference across lines is Not Associated with user.Phones TA(Line A), TB(Lines B1, B2) and T3(Lines C); TC is a phones which allows connected conference across lines. Observe All; GC1: A calls B1, GC2: B2 calls C		

Action	Events	Call info/Expected result
Do connected Conference Across Lines manually on Phone TB (which supports this feature) to conference GC1 and GC2	App, gets an PlatformExceptionImpl ("Terminal is restricted", CiscoJtapiException.CTIERR_DEVICE_RESTRICTED) when the add observer on TB, B1 and B2 GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	CiscoConferenceStartEv.getControllerAddress() returns B1CiscoConferenceStartEv.getControllerTerminalName() returns TB

Enhanced MWI Use Cases

Action	Result
Application calls CiscoAddress.setMessageSummary() to set voice and fax counts on a phone that supports the enhanced message waiting counts.	Phone displays updated voice and fax counts provided and also updates the MWI indicator accordingly. A successful response is returned.
Application calls CiscoAddress.setMessageSummary() to set voice and fax counts on a phone that does not support the enhanced message waiting counts.	Phone only updates the MWI indicator accordingly—no voice and fax counts are displayed on the phone. A successful response is returned
Application calls CiscoAddress.setMessageSummary() to set voice counts, but the “high priority” voice counts provided are bigger than “total” voice counts provided.	The request fails with the following error returned: INVALID_HIGH_PRIORITY_VOICE_COUNTS

Action	Result
Application calls CiscoAddress.setMessageSummary() to set fax counts, but the “total” fax counts provided is bigger than maximum size allowed.	The request fails with the following error returned: INVALID_TOTAL_FAX_COUNTS

Join Across Lines Enhancements

A, C, D, E and F are addresses on different terminals. B1 and B2 are addresses on the same terminal TermB.

A, B1 and C are in a conference call GC1 with B1 as the controller and connected to conference bridge Conference-1. B2, D and E are in conference call GC2 with D as controller and connected to bridge Conference-2.

Action	Events
Application conferences the two calls on B1 and B2 by invoking GC1.conference(GC2) to chain two conference call.	**Events to CallObserver of A, C and B1:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2Ev.getConferenceChain().getChainedConferenceCalls() will returnGC1 **Event for CallObserver at B2, D & E:** ConnDisconnectedEv B2 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection of Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 & Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 & GC2

Action	Events
Application invokes GC2.conference(GC1) to chain two conference calls.	**Event for CallObserver at B2, D & E:** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 **Events for CallObservers at A, B1 & C:** ConnDisconnectedEv B1 GC1 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will returnGC1

Action	Events
A, B1, C are in conference-1 (GC1), B1, D, E are in conference-2 (GC2), B2, F, G are in conference-3 (GC-3) Application completes conference at C by initiating GC1.conference(GC2, GC3) setting B1 as controller.	

Action	Events
	Event for CallObserver at A, B1 & C: TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEvTermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-3 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 & Conference-3 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 & GC3 **Event for CallObserver at B1, D & E:** ConnDisconnectedEv B1 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1-GC2 Ev.getConferenceChain().getChainedConferenceCalls() will returnGC2 **Event for CallObserver at B2, F & G:**

Action	Events
	ConnDisconnectedEv B2 GC3 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC3 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC3

Call Scenario: A, B1 and C are in conference call GC1 with B1 as controller. B2 is in call GC2 with D

Action	Events
Application sets the requestor as B2 and calls GC2.conference(GC1) getControllerAddress() returns B2. getOriginalControllerAddress() returns B1.	A CiscoConferenceStartEv CallCtlTermConnTalkingEvTermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
If application uses B1 as request controller in the above setup getControllerAddress() returns B1. getOriginalControllerAddress() returns B1.	Events are same as above

Swap or Cancel and Transfer or Conference Behavior Change

Use Case 1		
Connected Transfer on the phone which allows connected Transfer	GC1 & GC2 call will be created as normal.	
A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2	GC1: CallCtlTermConnHeldEv for TB	
User B presses transfer and user selects active call(A–>B call) from the phone UI	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
User B presses transfer again	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Use case 2		
Connected Transfer on phone with sharedline (A and A' are sharedlines) A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer and selects active calls (A–>B call), User B presses transfer again	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB| GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA' GC2: TermConnPassiveEvent for TA' GC2: CallCtlTermConnInUseEv for TA' GC1: TermConnDroppedEv for TA' GC1: CallCtlTermConnDroppedEv for TA' GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2:TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Use case 3		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer hard key User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns GC3

Use case 4a		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer hard key User press select active calls key. User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns null

Use case 4b		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer (or conference) hard key. User press select active calls key and also selects GC1 (A‡B call) User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns GC1

Use case 5		
Consult Transfer – Swap calls A calls B, B answers – GC1 B puts A–>B call on hold B setup consult Transfer to C, C answers – GC2 User B presses Swap key, User B presses transfer to complete the transfer	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL

Use case 6		
Consult Transfer – Swap/Cancel A calls B, B answers – GC1 A puts A–>B call on hold B setup consult Transfer to C, C answers – GC2 User B presses press Swap softkey, User B presses Cancel softkey	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB For TA (GC2), CallCtlTermConnHeldEv For TA (GC1), CallCtlTermConnTalkingEv GC1: CiscoCallFeatureCancelledEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 7		
Consultative Transfer Initiated from Phone, App sends SetupTransfer/Conference request – request fails A calls B, B answers – GC1 B setups transfer call to C B calls C, C answers – GC2 Application creates a new call and sends another consult() request	GC1 & GC2 call will be created as normal. Request will fail with PlatformException “CTIERR_CONSULTCALL_ALREADY_OUTSTANDING”	

Use case 8a		
Consult Transfer/Conference – Application Resumes primary call on phone which supports connected transfer/conference and sends another consult setup request GC1: A calls B GC2: B consults C Application resumes GC1 on TB Application creates another call and sends consult() request to call D; D answers	GC1 and GC2 will be created as normal For TB (GC2), CallCtlTermConnHeldEv For TB (GC1), CallCtlTermConnTalkingEv CiscoCallFeatureCancelledEv Consult call will go through and GC3 will be created as normal	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 8b		
Consult Transfer/Conference – Manually Resume primary call on phone which supports connected transfer/conference and then sends another consult setup request GC1: A calls B GC2: B consults C User manually resumes (SWAP) GC1 on B Application creates another call and sends consult() request to call D; D answers	GC1 and GC2 will be created as normal On Manual Resume or Swap, Consult Call will not be cancelled on the phone, nor will application get CiscoCallFeatureCancelledEv. When application tries to setup another consult, it will go through (GC3 will be created as normal) and it will cancel the existing consult call and application will get: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 9		
Connected ConferenceA (Phone which allows connected conference) calls B, B answer, B puts A onhold, B calls C, C answer B press “Conference” hardkey, and picks active call from UI, and selects A‡B call B press “Conference” again to complete connected conference	GC1 & GC2 call will be created as normal. C1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	

Use case 10		
Consult Conference from Phone, then Swap and complete conference through phone A calls B, B answerB setup conference to C, C answer B press “Swap” softkey A press “Conference”	GC1 & GC2 call will be created as normal. GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL

Use case 11		
Consult Conference from Phone and then Swap and Cancel conference thru phone A calls B, B answer B setup conference to C, C answer A press “Swap” key, and picks active call from UI, and selects A–>B call B press “Cancel”	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnTalkingEv for TB GC2: CallCtlTermConnHeldEv for TB GC1: CiscoCallFeatureCancelledEv(consultCall = GC2)	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelled Ev.getCall() returns GC1 CiscoCallFeatureCancelled Ev.getConsultCall() returns GC2

Use case 12		
Connected Conference Across Lines	Same as JAL scenario but we will have a temporary call GC3	

Use case 13		
Manual Consult followed by transfer complete by application GC1: A calls B1 GC2: B1 setups consult call to C manually over phone G1.transfer(GC2)	At Transfer: GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC1: CiscoCallFeatureCancelledEv GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEvent for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEv	

Use case 14		
Manual consult followed by conference complete by application GC1: A calls B1 GC2: B1 setups consult call to C manually over phone G1.conference(GC2)	At Conference: **GC1: CiscoCallFeatureCancelledEv** GC1: CiscoConferenceStartEv GC1: CiscoCallFeatureCancelledEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoConferenceEndEv	

Drop Any Party Use Cases

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 1 Application is observing A, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of C. Application invokes Connection.disconnect() on Connection of A.	InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 2 Application is observing C, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case3 Application is observing A and C. B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 4 Application is observing B, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv And B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 5 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE.
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 6 Application is observing A, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of C. Application invokes Connection.disconnect() on Connection of A.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 7 Application is observing C, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 8 Application is observing A and C. B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 9 Application is observing B, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 10 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE. A and A’ are shared line
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	Call info
Use Case 11 Application is observing A, B is conference controller. A, A' and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getPartyInfo() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) is dropped out of conference TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are A(abc) is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 12 Application is observing A', B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv . A'("xyz") is dropped out of conference... InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'("xyz") is dropped out of conference..	N, A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 13 Application is observing A and A'. B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A'(xyz) is dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) is dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) and A'(xyz)is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 14 Application is observing B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" No Events A'(xyz) is dropped out of conference. No Events A(abc) is dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEv B is disconnected from conference.g ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	Call info
Use Case 15 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A(xyz), dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc), dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is disconnected from conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.

- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE. A and A’ are shared line
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 16 Application is observing A, B is conference controller. A, A’ and B are in conference. Displayname for A is “abc”, and displayname for A’ is “xyz”	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of “abc” and “xyz” ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. No Events. A’(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc) is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 17 Application is observing A', B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. No Events. A(abc) is disconnected from conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'(xyz) dropped out of conference TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(xyz) is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 18 Application is observing A and A'. B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' No Events. A'(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(xyz) is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 19 Application is observing B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid B is disconnected from conference. No Events. A'(xyz) is disconnected from conference. No Events A(abc) dropped out of conference ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 20 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Display name for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames () at Connection of A. Application invokes Connection.disconnect () on Connection of B Application invokes Connection.disconnect (xyz) on Connection of A. Application invokes Connection.disconnect (abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A'(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

A, B, C are in conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns "True"	N..A
A, B, C are in conference. B drops from conference	Application invokes CiscoCall.isConferenceCall()	Interface Returns "False"	N..A

A, B, B’ are in conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A
A, B, B’ are in conference, B’ drops from conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns “False”	N..A
A, B, C are in conference. Applications opens provider, gets snapshot call event	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A
A, B, B’ are in conference. Applications opens provider, gets snapshot call event	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 21 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEV A, B C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 22 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, B, and C are dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 23 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A(xyz), dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc), dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, A' and B is disconnected from conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
	Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter "Advanced Ad Hoc Conference Enable" is set to TRUE.
- Cisco Unified Communications Manager service parameter "**Drop Ad Hoc Conference"** set "When controller leaves"

Scenario	Action	Result	CallInfo
Use Case 24 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference.	Application invokes Connection.getDisplayNames () at Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz"	N.A.

Scenario	Action	Result	CallInfo
Display name for A is “abc”, and displayname for A’ is “xyz”	Application invokes Connection.disconnect () on Connection of B Application invokes Connection.disconnect (xyz) on Connection of A. Application invokes Connection.disconnect (abc) on Connection of A.	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, A’ and B is dropped out of conference. TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ A’(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL	

Park Monitoring Support

Phone B—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Phone A—Future models.

Phone A'—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Park DN—P1, P2

Phone C—Cisco Unified IP Phone 7900 Series with SIP/SCCP

All the default values for the Park Monitoring Reversion timer and Park Monitoring Forward No reversion timers apply.

Use Case 1: Park Monitoring States

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 2 After step 1, Park Monitoring reversion timer expires after the configured time	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3 After step 1 or 2, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = RETRIEVED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 4 After step 1 or 2 above, B drops off the call invoking CiscoConnection.disconnect() on the connection of B.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = ABANDONED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 5 Consider Park Monitoring forward no retrieve destination on A is configured as F After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to F	**Events received at Call Observer on A, B** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 6 Consider Forward no retrieve destination on A is configured to self • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 7 Consider Forward no retrieve destination is not configured • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 2: Shared Line Scenario - Cisco Unified IP Phone Does Park

Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Consider Forward no retrieve destination is not configured, • Consider Park Monitoring Reversion timer and Park Monitoring Forward no reversion timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A **Note** all shared lines ring as is today	Reason = CiscoFeatureReason.PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 3: Shared Line Scenario - Cisco Unified IP Phone 7900 Series with SIP Does Park

Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A' answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Note** New event is not seen as Cisco Unified IP Phone 7900 Series does park	

Action	Result	Event/Call info
Step 2 Consider Park Reversion timer expires • Call is reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Note** All shared lines including the Cisco Unified IP Phone (future model) phone A receives the incoming call	Reason = CiscoFeatureReason.PARKREMINDER

Use Case 4: Use Case for Snap Shot Scenario

Initial scenario: Application has added Call Observer on A, B. Application has NOT added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 After step 1, application now adds Address Observer on A.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 3a After step 2, consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3b After step 1, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
Step 4 After step 3, application now adds Address Observer on A.	**New address event with park** state = RETRIEVED is not received at A, since the call is already retrieved	

Use Case 5: Park DN Is Monitored

Initial scenario: Application has added Call Observer on A, B. Application invokes registerFeature() API on Provider in order to monitor park DN P1. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Provider Observer Prov1** CiscoProvCallParkEv **Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 6: Query Number of Parked Calls

Initial scenario: Application has added Call Observer on A, B, C.

Action	Result	Event/Call info
Step 1 B calls A. A answers. Application invokes CiscoConnection.park() on connection on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 C calls A. A answers. Application invokes CiscoConnection.park() on connection on A for the second call on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
Step 3 Application invokes CiscoAddress.getAddress CallInfo(Term A) Application invokes CiscoAddress CallInfo.getNumParkedCalls()	CiscoAddressCallInfo is returned which includes information about number of parked calls getNumParkedCalls() returns 2	

Use Case 7: Filter Enabling or Disabling

Initial scenario: Application has added Call Observer on A, B. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Initially filter is disabled. • Application adds AddressObserver on A. • Application now invokes CiscoConnection.park() on connection on A. • Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event received as filter is disabled	

Action	Result	Event/Call info
Step 2 After step 1, Application enables filter via setCiscoAddrPark StatusEvFilter(true) and then by invoking CiscoAddress. setFilter(CiscoAddrEvFilter), for being able to receive the events.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 8: Filter Enabling or Disabling

Initial scenario: From the phone B calls A. A answers.(Call Observers are not added)

Action	Result	Event/Call info
Step 1 Initially filter is enabled. • Application adds AddressObserver on A. • Application now invokes park directly from the phone A. • Park reversion timer starts	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 9: Filter Enabling or Disabling

Initial scenario: From the phone B calls A. A answers.(Call Observers are not added)

Action	Result	Event/Call info
Step 1 Initially filter is disabled. • Application adds AddressObserve on A. • Application now invokes park directly from the phone A. • Park reversion timer starts. • Application now enables filter and invokes CiscoAddress. setFilter(CiscoAddr EvFilter)	**Events received at Address Observer on A** No event received yet, since filter is disabled **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Park reminder timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 10: Filter Enabling or Disabling

Initial Scenario : Initial scenario: Application has added Call Observer on A, B. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Initially all filters are disabled in CiscoAddEvFilter • Application adds AddressObserver on A. • Application now invokes CiscoConnection. park() on connection on A. • Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event received as filter is disabled	
Step 2 After step 1, Application invokes setCiscoAddrPark StatusEvFilter(true) but does not invoke CiscoAddress. setFilter (CiscoAddrEvFilter)	**Events received at Address Observer on A** No event received as the address filter is not set.	
Step 3 Now the application invokes setFilter (CiscoAddrEvFilter) on CiscoAddress	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_SNAPSSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Additional Use Cases for Park Monitoring

Phone B—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Phone A—Future models.

Phone A'—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Park DN—P1, P2

Phone C—Cisco Unified IP Phone 7900 Series with SIP/SCCP

All the default values for the Park Monitoring Reversion timer and Park Monitoring Forward No reversion timers apply.

1. Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 • Application invokes CiscoConnection.park() on connection on A. • Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 2 After step 1, Park Monitoring reversion timer expires after the configured time	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3 After step 1 or 2, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = RETRIEVED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 4 After step 1 or 2 above, B drops off the call invoking CiscoConnection.disconnect() on the connection of B.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = ABANDONED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 5 Consider Park Monitoring forward no retrieve destination on A is configured as F • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to F	**Events received at Call Observer on A, B** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 6 Consider Forward no retrieve destination on A is configured to self • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 7 Consider Forward no retrieve destination is not configured • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

2. Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A answers. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Consider Forward no retrieve destination is not configured, • Consider Park Monitoring Reversion timer and Park Monitoring Forward no reversion timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1	
	Events received at Address Observer on A CiscoAddrParkStatusEv **A** **Note** All shared lines ring as is today	Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

3. Initial scenario: Application has added Call Observer on A, B. Application has NOT added Address Observer on A. B calls A. A answers. Filter value has been set to ‘true’ through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 After step 1, application now adds Address Observer on A.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 3a After step 2, consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3b After step 1, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv BC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
Step 4 After step 3, application now adds Address Observer on A.	New address event with park state = RETRIEVED is not received at A, since the call is already retrieved	

4. Initial scenario: Application has added Call Observer on A, B, C. Filter value has been set to ‘true’ through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 B calls A. A answers. Application invokes CiscoConnection.park() on connection on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	
Step 2 C calls A. A answers. Application invokes CiscoConnection.park() on connection on A for the second call on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
Step 3 Application invokes CiscoAddress.getAddressCallInfo(Term A) Application invokes CiscoAddressCallInfo.getNumParkedCalls()	CiscoAddressCallInfo is returned which includes information about number of parked calls getNumParkedCalls() returns 2	

5. Use case to check for address event filter to control event notification- Filter value is set to ‘false’ through setCiscoAddrParkStatusEvFilter(). This is also the default value.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 By default the address event filter value is false. Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event notification since filter value is false	

6. Use case to check for address event filter to control event notification. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application enables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true). Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 After step 1 above, Application now disables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(false). Consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** No event notification as filter is disabled	
Step 3 After step 2 above, Application now enables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true).	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_SNAPSHOT park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

7. Use case to check the value of the filter set for the event CiscoAddrPArkrStatusEv.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application disables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(false) Application invokes the API getCiscoAddrParkStatusEvFilter() on CiscoAddrEvFilter.	The application receives the Boolean value ‘false’.	
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true) Application invokes the API getCiscoAddrParkStatusEvFilter on CiscoAddrEvFilter.	The Application receives the Boolean value ‘true’.	

8. Use case to check the notification of CiscoAddrIntercomInfoChangedEv and the value of the filter for the event, when the Intercom feature (target DN and/or intercom taget label) has not been changed.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 Application has set the filter value to 'false' through CiscoAddrEvFilter. setAddrIntercomInfo ChangedEvFilter(false) Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value 'false'.	
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrIntercomInfo ChangedEvFilter(true) Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A No events received as the intercom Feature is unchanged. The application receives the Boolean value 'true'.	

9. Use case to check the notification of CiscoAddrIntercomInfoChangedEv and the value of the filter for the event, when the Intercom feature (target DN and/or intercom taget label) has been changed.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 Application has set the filter value to 'false' through CiscoAddrEvFilter. setAddrIntercomInfo ChangedEvFilter(false) Application issues CiscoIntercomAddress.setIntercomTarget() on intercom address A Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value 'false'.	

Action	Result	Event/Call info
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrIntercomInfo ChangedEvFilter(true) Application issues CiscoIntercomAddress.setIntercomTarget() on intercom address A Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrIntercomInfoChangedEv **A** The application receives the Boolean value 'true'.	

10. Use case to check the notification of CiscoAddrIntercomInfoRestorationFailedEv and the value of the filter for this event.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 • Application has set the filter value to 'false' through CiscoAddrEvFilter. setCiscoAddrIntercomInfo RestorationEvFilter(false) • Application has set intercom target DN and label for intercom address A. Now CTI Manager goes outofservice, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN , label and UnicodeLabel to the set values, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI. • Applications invokes the API getCiscoAddrIntercomInfo RestorationEvFilter() on CiscoAddrEvFilter.	Events Received at Address Observer on A No notification as the filter is disabled. The Application receives a Boolean value 'false'	

Action	Result	Event/Call info
Step 2 • The application enables the filter through the API CiscoAddrEvFilter. setCiscoAddrIntercomInfo RestorationEvFilter(true) • Application has set intercom target DN and label for intercom address A. Now CTI Manager goes outofservice, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN , label and UnicodeLabel to the set values, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI. • Applications invokes the API getCiscoAddrIntercomInfo RestorationEvFilter() on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrIntercomInfoRestorationFailedEv **A** The Application receives a Boolean value ‘true’	

11. Use case to check the notification of CiscoAddrRecordingConfigChangedEv and the value of the filter for this event.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Recording Profile Configurations Settings have not been changed

Action	Result	Event/Call info
Step 1 • Application has set the filter value to ‘false’ through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter is set to default value (false) • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value ‘false’.	

Action	Result	Event/Call info
Step 2 • Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter(true) • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A No events as the Recording settings are unchanged. The application receives the Boolean value ‘true’.	

12. Use case to check the notification of CiscoAddrRecordingConfigChangedEv and the value of the filter for this event.

    Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 • Application has set the filter value to ‘false’ through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter (false)User changes the Recording Profile Configurations, through the Admin Pages. • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value ‘false’.	
Step 2 • Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter(true)User changes the Recording Profile Configurations, through the Admin Pages. • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrRecordingConfigChangedEv **A** The application receives the Boolean value ‘true’.	

Use Cases Related to DPark

Initial set up:

-Application has added call observer on B and A

-User has configured DPark DN **D**

- B is a future model Cisco Unified IP Phone

-A calls B. B answers with GCID GC1

Call Scenario	Expected behavior
Assisted DPark from a Cisco Unified IP Phone: • Cisco Unified IP Phone phone B (which is on active call with A) presses the pre-configured 'DPark BLF' button • The parked party A will be connected to D and hearMoH	When A(parked party) is connected to D, the following events are received **Events received at Call Observer on B, A** GC1 CallCtlTermConnHeldEv TB (CiscoFeatureReason. REASON_DPARK_CALLPARK) GC1 CiscoTermConnSelectChangedEv TB GC1 ConnUnknownEv B GC1 CallCtlConnUnknownEv B GC1 TermConnUnknownEv TB (CiscoFeatureReason. REASON_REFER)) GC1 ConnCreatedEv DGC1 ConnInProgressEv DGC1 CallCtlConnQueuedEv D (CiscoFeatureReason. REASON_REFER)) GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TBGC1 ConnDisconnectedEv BGC1 CallCtlConnDisconnectedEv B(CiscoFeatureReason. REASON_REFER))
DPark from Cisco Unified IP Phone • Cisco Unified IP Phone phone B (which is on active call with A) presses the 'Transfer' softkey • Parked party A is put on hold, and parker B dials **D** • Parker B is connected to D and hears MOH. • Parker B presses"Transfer" softkey again to complete the transfer of the parked party A to Dpark code D. • Parked party A is connected to D	No change in behavior. All events/reason remain the same as is today
DPark from JTAPI API • Application requests for a consult call from B, using the consult() API with destination address as DPark DN D. Say this call has GCID–GC2 • Application complete transfer, using the transfer() API GC1.transfer(GC2) • When transfer is completed A is connected to DPark DN	No change in behavior. All events/reason remain the same as is today

Call Scenario	Expected behavior
Unpark from JTAPI API • Consider application is observing C. • After step 3, application issues a request to unpark using the connect() API, with destination address as the prefix code followed by DPark code. • A is connected to C	No change in behavior. All events/reason remain the same as is today
Redirect to DPark DN via JTAPI API • B redirects to DPark code D, via the redirect() API with redirect destination as D.	No change in behavior. B is connected to DPark DN, but no park operation.

Logical Partitioning Feature Use Cases

Redirect from a Logical Partition (LP) Restricted Cluster

Terminal TA is configured with address A and is registered to a cluster which is configured with logical partition restrictions. Terminal TX is registered with address X which is configured to a cluster with no LP configuration.

Action	Result
X calls A. A redirects the call to a local PSTN number	PlatformException is thrown to redirect request. getErrorCode() on the exception returns CiscoJtapiException. REDIRECT_CALL_PARTITIONING_POLICY

Action	Result
A calls X (GC1) , X redirects the call to its local PSTN number	Originating cluster recognizes that the call is redirect to a PSTN and disconnects the call **Events delivered to Call Observer of A** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnDisconnected X GC1 CallCtlConnDisconnectedEv X GC1 CallInvalidEv

Call forward: Call to a address which is forward all to PSTM in GeoLocation with “disallowed” policy

Action	Result
setForward on A t o local PSTN Application is monitoring X. • X calls A using GC1.connect()	Connect() API succeeds but the call is dropped due to restrictions on A side. **Events delivered to call observer of X** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv X GC1 CallCtlTermConnDroppedEv TX GC1 ConnDisconnectedEv X GC1 CallCtlConnDisconnectedEv X GC1 ConnDisconnected A GC1 CallCtlConnDisconnectedEv A GC1 CallInvalidEv

Call Transfer: Transferring a call from different geo location to PSTN by controller in GeoLocation with “disallowed” policy

Action	Result
X calls A, A consults to PSTN number. Application is monitoring A. • A completes the transfer.	Platform exception is thrown to transfer() request. getErrorCode() returns CiscoJtapiException. TRANSFERFAILED

Call Conference: Conferencing a call from different location to PSTN by controller in GeoLocation with "disallowed" policy

Action	Result
X calls A, A consults to PSTN number. Application is monitoring A. • A completes the conference.	Platform exception is thrown to conference() request. getErrorCode() returns CiscoJtapiException. CTIERR_FEATURE_NOT_AVAILABLE.

Call Park / UnPark: Parking and un parking a PSTN call.

A and B are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as B

Action	Result
PSTN number calls A, A answers and parks the call. Application is monitoring A and B • B un-parks the call using unpark() API.	Call fails: ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED

Shared Lines

TermA and TermA' are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as TermA.

Action	Result
PSTN number calls A. Only TermA rings	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

Call Park Reversion with Shared Lines in Different Geographic Locations

TermA and TermA' are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as TermA.

Action	Result
PSTN number calls A, TermA answers and parks the call. After time out call is offered at TermA and not at TermA'	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

ComponentUpdater Enhancement Use Cases

Action	Result
Application calls ComponentUpdater(null)	Updater.log is creaated in the same directory
Application calls ComponentUpdater("C:\\temp\\")	Updater.log is created in c:\temp
Application calls ComponentUpdater(“readonlydirectory”) Application does not have write permission to Readonlydirectory	No log is created.

IPv6 Support

Use case for getIPAddressingMode()

Action	Result
Step 1 IP Adrressing mode for terminal A in Cisco Unified Communications Manager Admin pages is set as IPv4 Application invokes CiscoTerminal.getIPAddressingMode() on terminal A.	getIPAddressingMode() returns 0
Step 2 After step 1, the IP Addressing mode is changed from IPv4 to IPv6. User would be prompted to re set the device. Now application invokes CiscoTerminal.getIPAddressingMode() on terminal A.	getIPAddressingMode() returns 1(provided user had re-set the device)

Support for Cisco Unified IP Phone 6900 Series

Scenario / Description	Events to application
Application connects to CTIManager. • TermA is a Cisco Unified IP Phone 6921. • TermB is a Cisco Unified IP Phone 7931 with roll over mode. Admin now enables the new user role.	CiscoProviderCapabilityChangedEv – CiscoProvider.canObserverTerminalsWithRoleOver() returns true. CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() returns true. **Events to Provider Observer:** CiscoTermActivatedEv TermA CiscoTermA ctivatedEv TermB

Scenario / Description	Events to application
Admin removes the new user role.	CiscoProviderCapabilityChangedEv – CiscoProvider.canObserverlTerminalsWithRoleOver() returns false. CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() returns true. **Events to Provider Observer:** CiscoTermRestrictedEv TermACiscoTermRestrictedEv TermB
Term A is a Cisco Unfied IP 6900 series phone. Application does not have the new role enabled. Term A is in application control list Application adds observer on TermA	PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_DEVICE_RESTRICTED
Call Scenarios:	
Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1) Application issues consult request to Y (GC2). Call is created on A:P1	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEvGC2: ConnCreatedEv **A:P1** GC2: ConnConnectedEv **A:P1** GC2: CallCtlConnInitiatedEv **A:P1**
No roll over for incoming calls: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1). Applications calls A from B using connect API.	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv **B** GC2: ConnConnectedEv **B** GC2: CallCtlConnInitiatedEv **B** GC2: TermConnCreatedEv TernB GC2: CallCtlConnDialingEv B GC2: CallCtlConnEstablishedEv B GC2: ConnFailedEv B. getCiscoCause() returns CiscoCallEv.CAUSE_USERBUSY

Scenario / Description	Events to application
Roll over for Transfer and Conference (consult())only: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1). Applications calls connect() API from Adress A to Y. Similar exception will be seen for unPark(), startMonitor() requests.	Events delivered to Terminal Observer GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED
Only 1 address has callObserver: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on address A only. X calls A, call is answered Applications consults with Y using consult() API. On phone call consult call is created on A:P1	**Events delivered to CallObserver on A** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException is thrown. **getErrorDescription()** returns (“No callobserver on address A:P1). getErrorCode() returns **CiscoJtapiException.** ASSOCIATED_LINE_NOT_OPEN
Roll over to any line: In roll over, preference is giving to addresses with the same DN. If an address with the same DN is available it is choosen to roll over the consult call. Term A is configured with adress A, **B**, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1. Application adds callObserver on TermA X calls A, application answers the call. Applicaton consults with Y. The consult call is created on line3.	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv A:P1 GC2: ConnConnectedEv A:P1 GC2: CallCtlConnInitiatedEv A:P1

Scenario / Description	Events to application
Roll over to any line (same DN has another call): Term A is configured with adress A, **B**, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1. Application adds callObserver on TermA GC1: A:P1 calls Z, A:P1 holds the call GC2:X calls A, application answers the call Application consults GC2 to Y (GC3) Application completes the transfer	**Events delivered to Terminal Observer** GC2: ConnConnectedEv **A** GC2: CallCtlConnEstablishedEv **A** GC2: CallCtlTermConnTalkingEv TermA GC2: CallCtlTermConnHeldEv TermA GC3: CallActiveEv GC3: ConnCreatedEv B GC3: ConnConnectedEv B GC3: CallCtlConnInitiatedEv B … GC3: ConnCreatedEv Y .. GC3: CallCtlConnAlertingEv Y .. GC3: ConnConnectedEv Y GC3: CallCtlConnEstablishedEv Y CiscoTransferStartEv getTransferControllerAddress() returns A….CiscoTransferEndEv
Max call > 1: Term A is configured with adress A, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 3 and 2 on A. Application adds callObserver on TermA. X call A, A answers Application consults with Y Consult is setup on A (same line)	**Events delivered to Terminal Observer** … … GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv **A** GC2: ConnConnectedEv **A** GC2: CallCtlConnInitiatedEv **A**
Term A is configured with address A, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1/1 on A and A:P1. Application adds callObserver on TermA. A1 calls X. X answers the call – GC1 Y calls A. A answers the call and calls consult to Z	PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED

Terminal and Address Capability Settings Use Cases

Call Scenario	Expected behavior
Max Calls, busy trigger and Line Button: Address A is configured on TermA and TermA1. Line on TermA is configured with max calls 2 and line on TermA1 is configured with max calls 3. A on TermA is configured with busy trigger of 1 and A on TermA1 is configured with busy trigger of 2. A is on button 1 on TermA and on button 2 on TermA1.	A.getMaxCalls(TermA) returns 2 A.getMaxCalls(TermA1) returns 3 A.getMaxCalls(null) return 2 or 3 A.getBusyTrigger(TermA) returns 1 A.getBusyTrigger(TermA1) returns 2 A.getButtonPosition(TermA) returns 1 A.getButtonPosition(TermA1) returns 2
Voice Mail Pilot: Voice mail profile on address A is configured to point to pilot 2001 Voice mail profile on A is changed to point to to pilot 2002	A.getVoiceMailPilot() returns 2001 = CiscoAddrVoiceMailPilotChangedEv Ev.getAddress.getVoiceMailPilot() returns 2002
Labels: Address B on TermB is configured with ascii label = assciB and unicode label unicodeB.	B.getAsciiLabel(null) returns “asciiB” B.getUnicodeLabel(null) returns “unicodeB” B.getAsciiLabel(TermB) returns “asciiB” B.getUnicodeLabel(TermB) returns “unicodeB” B.getAsciiLabel(TermC) – throws Exception
IP Address TermC is registered in IPV4 mode only TermD is registered in IPV6 mode only TermE is registered in dual mode	TermC.getIPV4Address() returns a valid InetAddress TermC.getIPV6Address() returns null TermD.getIPV6Address() returns a valid InetAddress TermE.getIPV4Address() returns a valid InetAddress TermE.getIPV6Address() returns a valid InetAddress

Call Scenario	Expected behavior
Features: DTAL: TermF is Cisco Unified IP Phone model 7970 TermG is a CiscoRouteTerminal TermH is a CiscoMediaTerminal Join Across Lines: TermI has join across lines option enabled ConsultCallRollOver: TermJ is Cisco Unified IP Phone model 6921	TermF.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns true; TermF.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns true; TermG.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns false; TermG.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns false; TermH.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns true; TermH.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns false; TermI. canJoinAcrossLines (CiscoTerminal .APPLICATION) returns true; TermI. canJoinAcrossLines (CiscoTerminal .PHONE_USER) returns true; TermJ canJ ConsultCallRollOver (CiscoTerminal .APPLICATION) returns false; TermJ can ConsultCallRollOver (CiscoTerminal .PHONE_USER) returns false;
Provider has term1 and term2 in control list and both are registered to CUCM Application gets provider and registers for the register and unregister events. Provider.registerFeature(CiscoProvFeatureID. TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY Term1 unregisters	Provider Observer will get CiscoProvTermialUnRegisteredEv – getTerminal() returns Term1. Term1.isRegistered() returns false.
Term1 registers	CiscoProvTermialRegisteredEv – getTerminal() returns Term1. Term1.isRegistered() returns true.
Roll Over: TermK is Cisco Unified IP Phone model 7931 configured with rollover to any line. TermL is Cisco Unified IP Phone model 7940	TermK.getRollOverConfig() returns CiscoTerminal.ROLLOVER_ANY_DN. TermL.getRollOverConfig() returnd CiscoTerminal.NO_ROLLOVER.

All of the following use cases use the same basic configuration, unless specifically noted in the use case itsellf:

Pickup group1: N:P1 (pickup group number = N, partition = P1)

Pickup group2: M:P1

A, B and C are defined to be in pickup group1

D, E, and F are defined to be in pickup group2

Pickup group2 is subgroup for pickup group1

The following scenarios are basic use cases for the new Call Pickup APIs, and do not require an enumerated event list because of their simplicity. After these, two Call Pickup use cases will be presented so that you can see the new events in place. Interested parties should refer to the Unison JTAPI Interface Specification (EDCS-614242) for the full set of Pickup Use Cases. The Use Cases in that document will not have the new event, but after reading these use cases it should be readily apparent where they belong in the existing use cases.

Scenario	Action	Result
JTAPI Application is observing C	Application opens provider and issues CiscoAddress.getPickupGroupDN() and CiscoAddress.getPickupGroupPartition() at C	API returns N and P1 respectively
Cisco UCM service parameter AutoCallPickupEnabled is set to false.	Application opens provider and issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup()	API returns FALSE
Cisco UCM service parameter AutoCallPickupEnabled is set to true.	Application issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup()	API returns TRUE
Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	Application opens provider and issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup() ii). Now adminitrator sets AutoCallPickupEnabled service parameter to TRUE, iii). Application calls API CiscoProviderCapabilities.canAutoPickup()	i) API returns FALSE ii). CiscoProviderCapabilityChangedEv is delivered iii). API returns TRUE
JTAPI Application is observing C.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application opens provider and issues provider.unregisterPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. iv). A calls B	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Unregistration successful. iv). No Events for pickup alert.

Scenario	Action	Result
JTAPI Application is observing C. Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls starts ringing at C.
JTAPI Application is observing C. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call is answered at C.
JTAPI Application is observing C and D Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.groupPickup(Address of D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call starts ringing at D.
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTermina;.groupPickup(Address of D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call is answered at D.
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii).Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call starts ringing at D.

Scenario	Action	Result
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i)Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.otherPickup(Address of C) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at D
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B and then E calls C iii). Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 CiscoProvPickupCallAlertEvent with calling E, Called C, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at D. (Longest ringing call is picked up)
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Application will get PlatformException with error CTIERR_NO_CALLS_TO_PICKUP.
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.groupPickup(Address fo D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Request successful, new call created, but call gets disconnected with reason NORMAL.

Scenario	Action	Result
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Application will get PlatformException with error CTIERR_NO_CALLS_TO_PICKUP and cause NORMAL
JTAPI Application is observing C, D.	i). Application opens provider and issues provider. registerPickupAlert(K, P1) to register for pickup alert notification for call pickup group K:P1, where K is not a valid group number	i) PlatformException thrown with error CTIERR_INVALID_GROUP_NUMBER and cause NORMAL
	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). Application again issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1.	i). Registration successful ii.) Registration blocked at the JTAPI layer, and a InvalidStateException is thrown to the application with error CTIERR_ALREADY_REGISTERED and description “Pickup Group already registerred / observed”.
	i). Application opens provider and issues provider.deregisterPickupAlert(N, P1) to deregister for pickup alert notification for call pickup group N:P1, which was not registerred for previously.	i). InvalidStateException thrown to application with error CTIERR_REGISTRATION_NOT_FOUND and description, “Pickup group is not registerred with this provider”.
JTAPI Application is observing J. Cisco UCM service parameter “AutoCallPickupEnabled” is set to true. J is an address in two partitions, P1 and P2. J belongs to the Pickup Group with number N.	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.pickup(Address of J in P1) at terminal of J in P1	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at J in P1.

Scenario	Action	Result
JTAPI Application is observing J. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. J is an address in two partitions, P1 and P2. J belongs to the Pickup Group with number N..	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.pickup(Address of J in P2) at terminal of J in P2	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at J in P2.
JTAPI Application is observing K. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. K is an address in partition P3, and belongs to Pickup Group 3, in partiton P3. Pickup Group 3 has the same number as Pickup Group 1, N. CSS of K is configured to check P3:P1.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.groupPickup(Address of K in P3, N) at terminal of K in P3.	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii.) InvalidStateException thrown to application with error CTIERR_NO_CALLS_TO_PICKUPand description, "There are no calls to pickup".
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) The CUCM crashes / goes offline, and CTI failsover to the the secondary CUCM server.	i). Registration successful ii.) The JTAPI layer sense the failover and re-registers for the pickup groups that it was registers for (N, P1).
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) An administrator logs into the CUCM Admin page and changes the DN for the Pickup Group.i).	i) Registration successful ii.) Application recieves a ProviderPickupNotificationRegistrationClosedEvent, with reason = CTIERR_PICKUPGROUP_CHANGED
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) An administratior logs in to the CUCM Admin page and changes the "Auto Pickup Enabled" service parameter.	i.) Registration successful ii.) Application receives a CiscoProviderCapabilityChangedEv. CiscoProviderCapabilities. canAutoPickup() will be changed to reflect the new setting.

Scenario	Action	Result
JTAPI Application is observing J. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. J is an address in partition P3. J belongs to the Pickup Group with number N, in P1. J's CSS is not configured to check P1.	i) Application opens provider and issues provider.registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B iii.) Application issues CiscoTerminal.pickup(Address of J in P3) at terminal of J in P3.	i) Registration successful ii.) After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii.) PlatformException is thrown with reason = CTIERR_TEMPORARY_FAILURE and description "Temporary Failure".

Full-Event Use Case 1: (Observing All Devices): Auto-Pickup Disabled

Action	Call event	Call ID / Info
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes Terminal.pickup(Address of C)	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv CiscoProvPickupCallAlertEvent GC2-CallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B Calling A, Called B, PUG Number N, PUG partition P1 CCalling C, CCalled: NONE LRP: NONE REASON_NORMAL

Action	Call event	Call ID / Info
Call 2 gets dropped / invalidated C gets a connection on Call 1 B is dropped from Call 1 C is ringing C is on call with A	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	REASON_CALLPICKUP CCalling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP CCalling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP REASON_NORMAL REASON_NORMAL

Full-Event Use Case 2 (Observing All Devices): Auto-Pickup Enabled

Actions	Events	Call info (GCID: Info)
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing … pause for Pickup Group's delay Provider recieves Pickup Alert C invokes Terminal.pickup(Address of C)	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE

Actions	Events	Call info (GCID: Info)
Old Conn for C is dropped B is dropped / cleaned up C's connection on Call 1 is established	GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallCtlConnEstablishedEv-C	REASON_CALLPICKUP CCalling: A, CCalled: C LRP: NONE REASON_CALLPICKUP CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL REASON_CALLPICKUP CCalling: A, CCalled: C LRP: B REASON_NORMAL

Full-Event Use Case 3 (Observing All Devices): Group Pickup, Auto-Pickup Enabled

Action	Call event	Call Id/Info
provider. registerPickupAlert(N, P1) [Basic call happens, see case 1] … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes groupPickup(Address of C, N) C is dialing the PU Number C is added to the original call Pickup added to original call	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP:B

Action	Call event	Call Id/Info
Pickup # is removed Call 2 C is dropped from Call 2 Pickup # is removed Call 1 B is dropped / invalidated	GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	REASON_CALLPICKUP, LRP: PU CCalling: C, CCalled: PU REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP

As you can see, there are only a handful of changes for this Group Pickup case, and they all directly relate to the extra required step of dialing the Pickup Number. A similar test was run with Auto-Pickup disabled:

Full-Event Use Case 4 (Observing All Devices): Group Pickup, Auto-Pickup Disabled

Action	Events	Call info
provider. registerPickupAlert(N, P1) [Basic call happens, see Case 1] … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C is dialing the PU number PU is removed from Call 2 C is removed from Call 2 Call 2 is destroyed	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP

Action	Events	Call info
C gets a connection on Call 1 B is dropped from Call 1 C is ringing C picks up	GC1-ConnCreatedEvent[ADDRS] GC1-ConnInprogressEvent GC1-CallCtlConnOfferedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent GC1-CallCtlConnDisconnectedEv GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Observing Only Device C: Auto-Pickup Enabled

Actions	Call events	Call IDs/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes pickup(Address of C) at Terminal for C C is connected to Call 1 C is dropped from Call 2 Call 2 is invalidated / cleared A and C are connected on Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-CallCtlConnEstablishedEv-C	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REAON_CALLPICKUP CCalling A, CCalled: NONE LRP: NONE CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_CALLPICKUP REASON_CALLPICKUP CCalling A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

Observing Only Device C: Auto-Pickup Disabled

Actions	Events	Call events\ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes pickup(Address of C) at Terminal for C Call 2 is destroyed C is added to Call 1, but does not pick up	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP
C is ringing	GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	REASON_NORMAL
C picks up, and is connected to Call 1	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Observing Only Device C: Group Pickup, Auto-Pickup Enabled

Actions	Call event	Call ID/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C dials the Pickup Number C is added to Call 1 PU is added to Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP CCalling: A, C Called: C CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP

Actions	Call event	Call ID/ Call info
PU # is removed from Call 2 C is removed from Call 2 Call 2 I invalidated / cleared C is connected to Call 1 PU is removed from Call 1	GC2-ConnDisconnectedEvent-PUv GC2-CallCtlTermConnDroppedEvvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEnde GC2-ConnDisconnectedE GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEdEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-PU GC1-CallCtlConnEstablishedEv-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU	CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C REASON_CALLPICKUP CCalling: A, CCalled: C REASON_CALLPICKUP

Observing Only Device C: Group Pickup, Auto-Pickup Disable

Actions	Call event	Call ID/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C dials the PU Number PU is dropped from Call 2 C is dropped from from Call 2 Call 2 is destroyed C is added to Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC1-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_CALLPICKUP REASON_NOTMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP

Actions	Call event	Call ID/ Call info
C is ringing C is connected to A	GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	REASON_NORMAL

Invoking Pickup on a Ringing Shared Line (CSCsy30964)

This is an odd scenario that normal applications will probably not see.

A calls a shared line DN (B), that has SLT1 and SLT2 on it. B is in pickup group N, P1.

B and B’ ring, and instead of invoking answer(), the application invokes pickup() on B’.

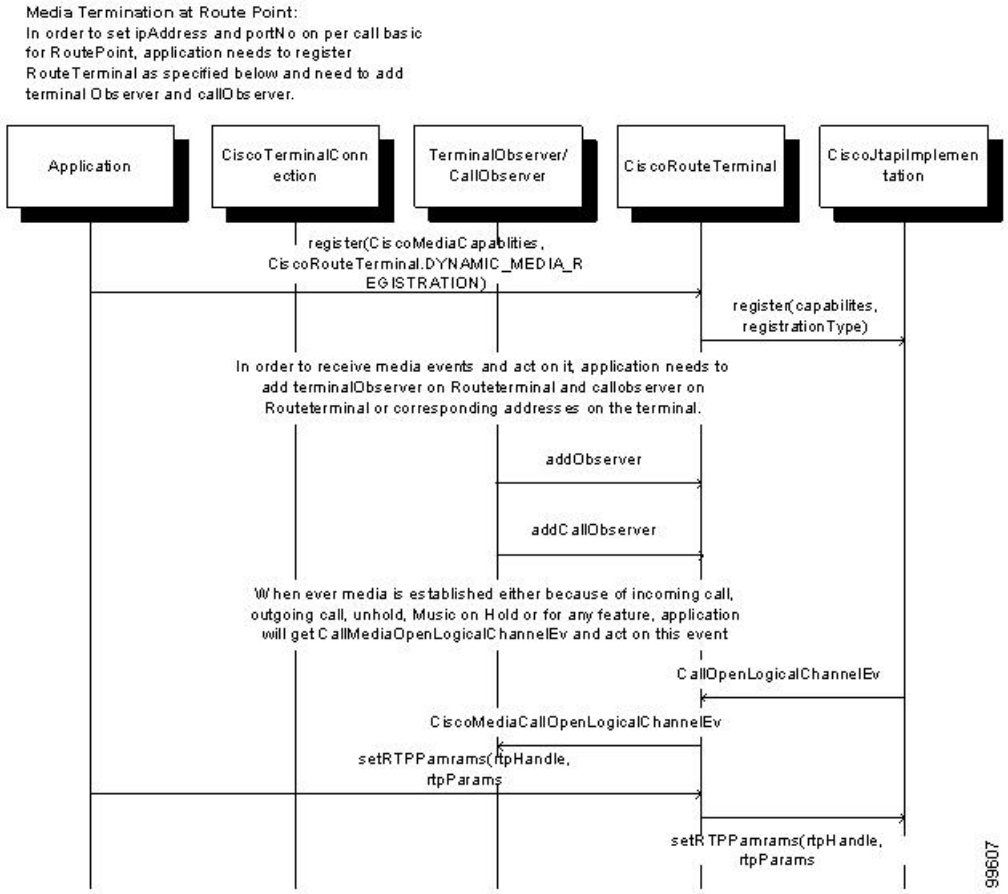
Action	Call event	Call ID/Info
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing on both shared lines	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnInprogressEvent GC1-ConnAlertingEvent GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1- CallCtlTermConnRingingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B

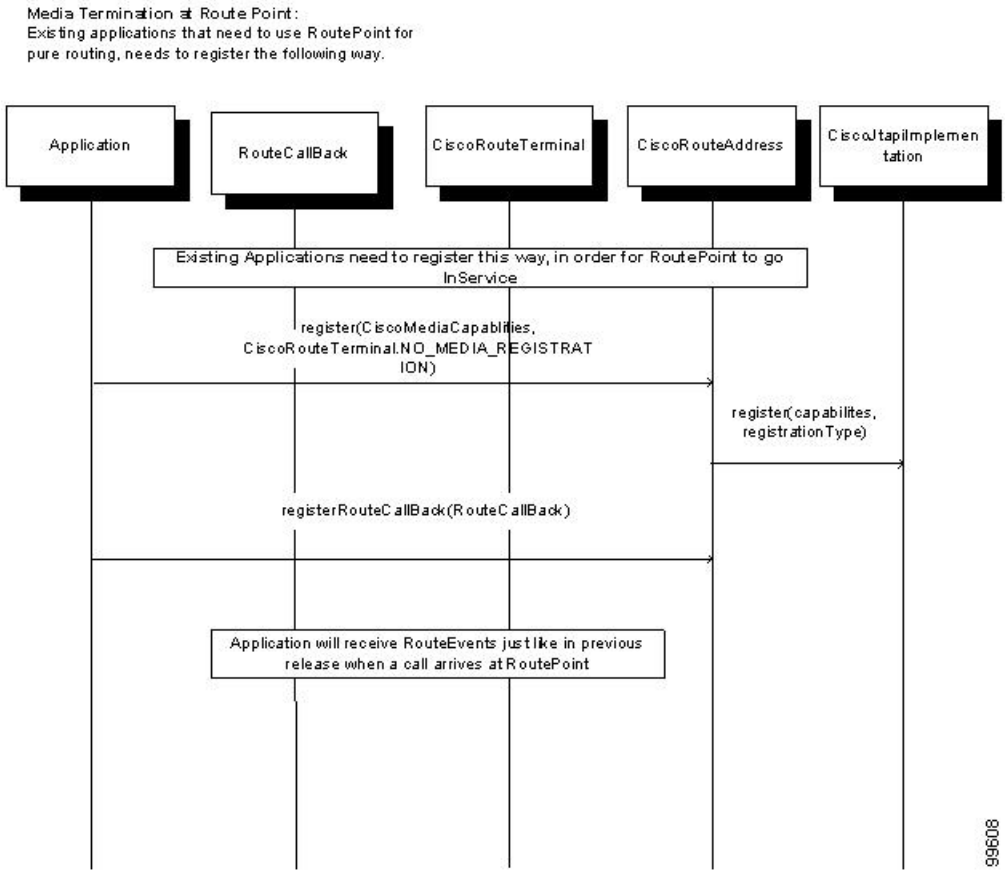
Action	Call event	Call ID/Info
… pause for Pickup Group's delay Provider receives Pickup Alert Application invokes Terminal.pickup(Address of B) on Terminal of SLT2 Shared line gets a connection on GC1 GC2 gets cleaned up SLT1 on Call 1 goes passive/bridged SLT2 on Call2 goes active SLT2 is talking	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnCreatedEvent GC2- TermConnPassiveEvent GC2-CallCtlTermConnInUseEv GC2-CiscoCallChangedEv GC1-ConnConnectedEvent GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-CiscoCallChangedEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent GC2-CallCtlConnDisconnectedEv GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnPassiveEvent GC1-CallCtlTermConnBridgedEv GC1-CallCtlConnEstablishedEv GC1-CallCtlTermConnInUseEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	Calling A, Called B, PUG Number N, PUG partition P1 CCalling C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling A, CCalled: B Calling: A, Called: B, LRP: B REASON_CALLPICKUP

Media Termination at Route Point

The following diagrams illustrate the message flows for Media Termination at Route Point.

Message Sequence Charts





Mobility Interaction Support

Pre-conditions to mobility interaction use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Enable "Route calls to all remote destinations" checkbox for CTIRD1 and CTIRD3.
- CTIRD1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "C1_CTIRD1_RDD3", Number: "339007")
- CTIRD3 associated to user "Mobility3", dn = 9202
    - Remote destination 1 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- RDP1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "C1_CTIRD1_RDP1", Number: "334007")
- RDP3 associated to user "Mobility3", dn = 9202
    - Remote destination 1 (Name: "C1_CTIRD3_RDP1", Number: "334003")
    - Remote destination 2 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- Device A (IP Phone - Name: "SEP2401C7824EA3", Line A1 (dn: 9000))

- User1 has in its control list: Device A, CTIRD1 and CTIRD3. All devices and lines are observed.

Table 279: Call to CTI RD When App Not Active with Unique RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
A1 calls CTIRD1 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303"). Call is answered at the RD of RDP1 (dn = 4007).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 *Call is offered to the remote destination of RDP1 after delay. Call is not offered to CTIRD1 or to the remote destination of CTIRD1 (dn = 9007).	

Table 280: Call to CTI RD When App Active with Unique RD with Answer on RD of RDP

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD1 to be 339007. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339007", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339007" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1	
Call is answered at the RD of RDP1 (dn = 4007).	GC1: CallCtlTermConnRingingEv CTIRD1 *Call is offered to CTIRD1 and to the RD of CTIRD1 without delay. Call is offered to RD of RDP1 after delay. GC1: TermConnDroppedEv CTIRD1 GC1: CallCtlTermConnDroppedEv CTIRD1	

Table 281: Call to CTI RD When App Active with Unique RD with Answer on RD of CTI RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD1 to be 339007. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339007", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339007" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 *Call is offered to CTIRD1 and to the RD of CTIRD1 without delay. Call is offered to RD of RDP1 after delay.	
Call is answered at the RD of CTIRD1 (dn = 9007).	GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1	

Table 282: Call to CTI RD When App Not Active with Shared and Unique RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
A1 calls CTIRD3 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202"). Call is answered at the RD of RDP3 (dn = 4003).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 *Call is offered to the remote destination of RDP3 after delay. Since 339006 is shared between CTIRD3 and RDP3, call is only seen at 339006 after delay.	

Table 283: Call to CTI RD When App Active with Shared and Unique RD with Answer on RD of CTI RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
App sets active rd of CTIRD3 to be 339006. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339006", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339006" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.

Action	Events	Call Info
A1 calls CTIRD3 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 *Call is offered to CTIRD3 and to the RD of CTIRD3 (which is shared with RDP3) without delay. Call is offered to the unique RD of RDP3 after delay.	
Call is answered at the RD of CTIRD3 (dn = 9006). This is the RD that is shared between CTIRD3 and RDP3.	GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	

Table 284: Call to CTI RD When App Active with Shared and Unique RD with Answer on RD of RDP

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD3 to be 339006. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339006", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339006" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD3 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 *Call is offered to CTIRD3 and to the RD of CTIRD3 (which is shared with RDP3) without delay. Call is offered to the unique RD of RDP3 after delay.	
Call is answered at the RD of RDP3 (dn = 4003).	GC1: TermConnDroppedEv CTIRD3 GC1: CallCtlTermConnDroppedEv CTIRD3	

Modifying Calling Number

The following scenario illustrates the message flows for Modifying Calling Number.

Scenario One

The application controls the device Route Point (RP) and registers RP .

A and B are PNO and appear within the Cisco Unified Communications Manager cluster.

A calls RP.

Call arrives at RP

Action	Event	Fields
Call Arrives at RP	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A)
Application invokes selectRoute(routeselected[], callingsearchspace, modifiyingcallingnumber[]) where routeSelected[] = C callingSearchSpace = CiscoRouteSession.DEFAULT_SEARCH_SPACE	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A) getRouteUsed () = C
Application invokes endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_END getRouteAddress () = RP

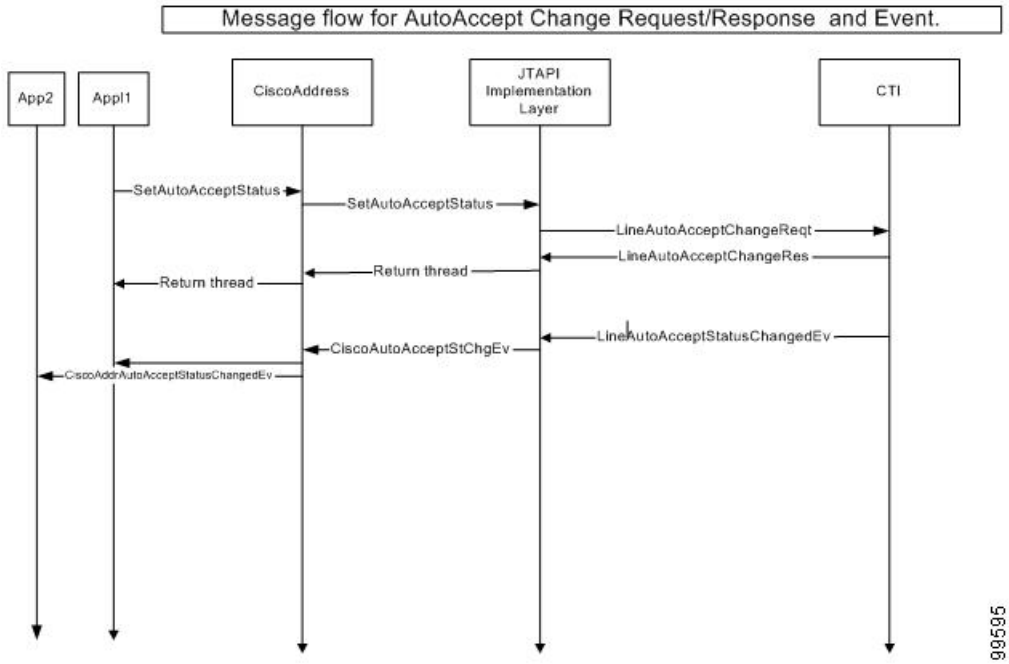
Scenario Two

The application is controls A and B.

A calls RP, which selects Route call to B with modified calling number as M.

Action	Event	Fields
A calls RP, which is not in controlled list.	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CallCtlConnEstablishedEv A ConnCreatedEv RP ConnInProgressEv RP CallCtlConnOfferedEv RP	getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = getModifiedCallingAddress() = A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = getModifiedCallingAddress() = A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = A getModifiedCalledAddress() = B
Another application controls the RP selectRoute to B with modifying calling number as M.	NEW META EVENT_____ META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnDisconnectedEv RP CallCtlConnDisconnectedEv RP NEW META EVENT_____ META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B

AutoAccept for CTIPort and RoutePoint



Silent Monitoring Use Cases

A and TA are address and terminal of monitor target or recording initiator

B and TB are address and terminal of monitor initiator.

Scenario One

Administrator enables monitoring capability for the user.

Action	Events	Call info
	CiscoProviderCapabilityChangedEv hasMonitorCapabilityChanged() on this event returns true hasRecordingCapabilityChanged() returns true	NA
ciscoProvider.getCapabilities().canMonitor()	JTAPI returns true	NA

Scenario Two

Start and Stop monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. Application has monitoring capability enabled.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A or GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, terminal name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorInitiatorInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A GC2: Calling: B Called: A LRP: null Current calling: B Current called: A

Action	Events	Call info
A puts the call on hold A resumes the call B calls drop on GC2 to stop monitoring	GC2: CiscoRTPOutputStoppedEv GC1: CiscoRTPOutputStoppedEv **GC1: CallCtlTermConnHeldEv** TA **GC2**:CiscoRTPInputStoppedEv **GC1**: CiscoRTPInputStoppedEv **GC1**: CiscoRTPOutputStartedEv GC2: CiscoRTPOutputStartedEv GC1: CallCtlTermConnTalking TA **GC2**:CiscoRTPInputStartedEv **GC1:** CiscoRTPInputStartedEv GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnEv A GC1: CiscoTermConnMonitorEndEv TA GC2: ConnDisconnEv B GC2: CallInvalidEv	

Scenario Three

Monitor initiator transfers the call to Y. A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. application has monitoring capability enabled. B transfers the call to Y.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 and ci2 giving CI1, A and TermA from GC1 Or using the teminalconnection of A.	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: ConnConnectedE**v for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A or GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL Monitor_TARGET address:A, device name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv TA** **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_NORMAL address:B, device name: TB rtphandle = ci2	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Action	Events	Call info
B consults Y using GC3 and completes transfer. Call observer on Y would see	GC3**:** CallActiveEv GC3: ConnConnectedEv B GC3: CallCtlTermConnTalkingEv TB GC3: ConnConnectedEv Y CiscoTransferStartEv(GC3->GC2) GC3: ConnDisconnectedEv Y GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:Y, device name: TY GC3: ConnDisconnectedEv B GC3: CallCtlTermConnDroppedEv TB GC3: CallInvalidEv GC2: CallCtlTermConnDroppedEv TB …. GC2: CallInvalidEv CiscoTransferEndEv GC3: CallActive **GC3:** CallCtlTermConnRingingEv TY GC3: CallCtlTermConnTalkingEv TY CiscoTransferStartEv CiscoCallChangedEv GC3->GC2 GC2: ConnConnectedEv Y **GC2**: ConnConnectedEv B GC2: CiscoTermConnMonitorTargetInfoEv TY Cause: CAUSE_NORMAL address:A, device name: TA GC3: ConnDisconnectedEv Y **GC3:** CallCtlTermConnDroppedEv TY GC3: CallInvalidEV GC2: ConnConnectedEv X GC2: ConnDisconnectedEv A CiscoTransferEndEv (**CiscoTermConnMonitorInitiatorInfoEv** on GC1 is independent of transfer events on GC3 and GC2 and can be delivered at any time before or after end event.)	GC1: Calling: X Called: A LRP: A Current calling: X Current called: Y

Scenario Four

Monitoring a barged call: A and A' are shared lines. Caller calls A, A answers the call. A' barge into the call. B calls start monitor.

Action	Events	Call info
A answers GC1 A' barges the call B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv GC1: CallCtlTermConnBridgedEv TermA' GC1: **GC1:CallCrlTermConnTalkingEv** TA' Exception is thrown to startMonitor request.	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Five

Monitor and recording: A is monitor target and has auto recording configured. B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. Application has monitoring capability enabled.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv GC1: CiscoTermConnRecordingStartEv TA **GC1: CiscoTermConnRecordingTargetInfoEv TA** CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A on GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, device name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorTargetInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Six

Observing remote in use shared line in Monitoring: A and A' are shared lines. Caller calls A, A answers the call. B calls start monitor. Application has call observer on A' only. B initiates monitor request for connected call on A. No start events are delivered to call observer of A'.

Action	Events	Call info
A answers GC1 and B initiates monitor A puts the call on HOLD A' answers the call B drops the call and initiates start monitor using GC3 giving the terminal connection of A' in monitor request.	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A' Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A' Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA' Cause: CAUSE_NORMAL GC1: CallCtlTermConnBridgedEv TermA' Cause: CAUSE_NORMAL **GC1: CallCrlTermConnHeldEv TA**' **GC1:CallCrlTermConnTalkingEv TA**' GC1: CiscoTermConnMonitorStartEv TA' **GC1: CiscoTermConnMonitorInitiatorInfoEv TA'** Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Seven

Snap Shot events for Monitor and recording: A is monitor target and has auto recording configured. B is monitor initiator. X calls A, A answers the call GC1 (ci1), B calls start monitor using GC2. Another application adds call observer on A after monitoring and recording sessions are established.

Action	Events	Call info
	CallActiveEv for callID = GC1 Cause: CAUSE_SNAPSHOT **GC1:ConnCreatedEv** for A Cause: CAUSE_SNAPSHOT **GC1:ConnConnectedEv** for A Cause: CAUSE_SNAPSHOT GC1: ConnConnectedEv X … **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingTargetInfoEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_SNAPSHOT address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Secured Monitoring Use Cases

Monitoring Use Cases

Scenario	Expected Result	Info
Scenario 1 1. Supervisor and agent's device security mode is encrypted 2. Customer and Agent are in a secured call. 3. Supervisor initiates a monitoring request.	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 TargetAddress = A TargetTerminal = TermA targetCall = GC1
4. The Agent Customer call is dropped.	GC1: CiscoTermConnMonitoringEndEv TermA … GC1: CallInvalidEv … GC2: CallInvalidEv	

Scenario	Expected Result	Info
Scenario 2 1. Supervisor and agent's device security mode is encrypted 2. Customer and Agent are in a non-secured call. 3. Supervisor initiates a monitoring request 4. Supervisor stops monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1

Scenario	Expected Result	Info
Scenario 3 1. Agent's Device is non-secured and supervisor is secured 2. Customer and Agent are in a non secured call 3. Supervisor initiates a monitoring request 4. Supervisor stops Monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
Scenario 4 1. Agent's device is secured and supervisor is non-secured 2. Customer and agent are in a secured/non-secured call 3. Supervisor initiates a monitoring request	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S .. GC2: CallCtlConnEstablished TermS GC2: ConnFailedEv S2 GC2: CallCtlConnFailedEv S2 PriviledgeViolationException :CTIERR_SECURITY_CAPABILITY_MISMATCH (as supervisor's does not meet the security capabilities of the Agent)	Cause: BCNAUTHORIZED

Scenario	Expected Result	Info
Scenario 5 1. Both supervisor and agent are non-secured 2. Customer and agent are in a non-secured call 3. Supervisor initiates a monitoring request 4. Supervisor stops monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: connCreatedEv A .. GC2: CallCtlConnEstablishedEv termA GC2: CiscoTermConnMonitoringTargetInfoEv GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
Scenario 6 1. Supervisor1 and agent are secured, supervisor2 is non-secured. 2. Customer and Agent are in a secured/non-secured call 3. A secured monitoring session is in progress with supervisor1.	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitoringInitiatorInfoEv termA GC2 : ConnCreatedEv A .. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitoringTargetInfoEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1 cause : CAUSE_BCNAUTHORISED

Scenario	Expected Result	Info
4. Supervisor1 transfers the monitoring call to supervisor2	CiscoTransferStartEv ….. CiscoTransferEndEv The monitoring session is torn down (since supervisor2 does not meet the security capabilities of the agent) GC2: ConnFailedEv S2 GC2: CallCtlConnFailedEv S2 GC1: CiscoTermConnMonitorEndEv Events Received on Supervisor1 CiscoAddrMonitoringTerminatedEv	TransactionID : xxxx AgentAddr = A AgentDevice = termA AgentCID = agentCI in GC1 Supervisor1 Device Name = TermS1 Cause = BCNAuthorised
Scenario 7 1. Supervisor1 and agent are secured, supervisor2 is secured. 2. Customer and Agent are in a secured/non-secured call 3. A secured monitoring session is in progress with supervisor1.	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitoringInitiatorInfoEv termA GC2 : ConnCreatedEv A .. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitoringTargetInfoEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
4. Supervisor1 transfers the monitoring call to supervisor2	CiscoTransferStarteEv …. CiscoTransferEndEv GC2: ConnCreatedEv S2 ….. GC2: CallCtlConnEstablishedEv TermS2 GC2: CiscoTermConnMonitoringTargetInfoEv	Targetaddress = A TargetTerminal = TermATarget call = GC1

Monitoring and Recording Use Cases

Scenario	Expected Result
Scenario 1 1. Agent is non-secured, Supervisor and recorder are encrypted 2. Agent is in a non-secured call with customer 3. Supervisor monitors the call in non-secured mode	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS
4. Application requests for recording the call at Supervisor	CiscoTermConnRecordingStartEv CallCtlTermConnTalkingEv TermS The recording is done in a secured mode as Both the supervisor and recorder are secured, but the overall call security is non-secured.

Native Queuing

1. Hunt Pilot HP has one member B.
2. Check box for “Queue Calls” is checked.
3. “Maximum In Queue timer” is set as 60 seconds.
4. “Destination When Maximum Wait Time is Met” is set as C
5. Queue depth is set to 1
6. “Destination When Queue is Full” is set to C

Queuing of Call

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 ConnInProgessEv HP GC2 CallCtlConnQueuedEv HP GC2 CallCtlConnDisconenctedEv HP GC2 ConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.INTERNAL Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT

De-queuing of a call

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 ConnInProgessEv HP GC2 CallCtlConnQueuedEv HP GC2 CallCtlConnDisconenctedEv HP GC2 ConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.INTERNAL Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT

Action	Event	Call info
B disconnects the call	GC1 ConnDisconnectedEv HP GC1 CallCtlConnDisconnectedEv HP GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconenctedEv B GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconenctedEv A GC1 CallInvalidEv GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertinEv B GC2 CallCtlConnAleringEv B GC2 TermConnCreatedEv Term B Gc2 termConnRingingEv TermB GC2 CallCtlTermConnRingingEv TermB GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = B Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
B Answers	GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnActiveEv Term B GC2 TermConnCtalkingEv TermB	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = B Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP

Maximum In-Queue Timer Expires

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Action	Event	Call info
After 60 seconds	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 CallCtltermConnRingingEv termC GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = C Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = C Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Maximum In-Queue Timer Expires with Destination as Another HP Whose Member E Is Free

All members of HP are busy. The destination when queue timer expires is HP2 whose members E is free

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
After 60s call gets offered on HP2	GC2 ConnCreatedEv E GC2 ConnInProgressEv E GC2 CallCtlConnOfferedEv E GC2 CiscoHuntConnCreatedEv HP2 GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E Gc2 termConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE GC2 ConnConnectedEv HP2 GC2 CallCtlConnEstablishedEv HP2 GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
E answers	GC2 ConnConnectedEv E GC2 CallCtlConnEstablishedEv E GC2 TermConnActiveEv Term E GC2 TermConnCtalkingEv TermE	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP

Maximum In-Queue Timer Expires with Destination as Another HP Whose Members Are Busy

All members of HP are busy. The destination when queue timer expires is HP2 whose members (E) is also busy

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()). getAddress()). getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
E becomes Free after >60s	GC2 ConnCreatedEv E GC2 ConnInProgressEv E GC2 CallCtlConnOfferedEv E GC2 CiscoHuntConnCreatedEv HP2 GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E Gc2 termConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE GC2 ConnConnectedEv HP2 GC2 CallCtlConnEstablishedEv HP2 GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING Ev.getConnection()). getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()). getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING (AddressImpl)(ConnectionImpl) ((ConnEvImpl) Ev.getConnection()). getAddress()).getType() = CiscoAddress.INTERNAL
E answers	GC2 ConnConnectedEv E GC2 CallCtlConnEstablishedEv E GC2 TermConnActiveEv Term E GC2 TermConnCtalkingEv TermE	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP

Queue Is Full

Action	Event	Fields
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Fields
E calls HP Call gets offered on C	GC3 CallActiveEv GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 CallCtlConnEstablishedEv E GC3 TermConnCreatedEv E GC3 CallCtlTermConnTalkingEv TermE GC3 CiscoHuntConnCreatedEv HP GC3 ConnCreatedEv C GC3 CallCtlCallOfferedEv C GC3 ConnAlertinEv C GC3 CallCtlConnAleringEv C GC3 TermConnCreatedEv Term C Gc3 termConnRingingEv TermC GC3 CallCtlTermConnRingingEv TermC GC3 CallCtlConnDisconnectedEv HP GC3 ConnCisconnectedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_BUSY Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_BUSY
C answers	GC3 CallCtlConnEstablishedEv C GC3 CallCtlTermConnTalkingEv termC	Call.getCalledAddress() = HP Call.getCallingAddress() = E Call.getCurrentCalledAddress() = C Call.getCurrentCallingAddress() = E Call.getModifiedCallingAddress() = E Call.getModifiedCalledAddress() = C Call.getLastRedirectedAddress() = HP

When Disconnect Is Selected for Queue Full

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
E calls HP Call fails	GC3 CallActiveEv GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 CallCtlConnEstablishedEv E GC3 TermConnCreatedEv E GC3 CallCtlTermConnTalkingEv TermE GC3 CiscoHuntConnCreatedEv HP GC3 ConnFaileEv E GC3 CallCtlConnFailedEv E Gc3 ConnDisconenctedEv HP Gc3 ConnDisconnectedEv E Gc3 CallCtlConnDisconenctedEv E Gc3 CallInvalidEv	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_NORMAL Cause = UserBusy

Same result as above is observed when "Disconnect" is selected for MaxQueueTime and Agents Not Logged in configs and those conditions get hit.

No Agents Are Logged In

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC3 ConnCreatedEv C GC3 CallCtlCallOfferedEv C GC3 ConnAlertinEv C GC3 CallCtlConnAleringEv C GC3 TermConnCreatedEv Term C Gc3 termConnRingingEv TermC GC3 CallCtlTermConnRingingEv TermC GC3 CallCtlConnDisconnectedEv HP GC3 ConnCisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_UNAVAILABLE Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_UNAVAILABLE Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT
C answers the call	GC1 CallCtlConnEstablishedEv C GC1 CallCtlTermConnTalkingEv termC	

Caller Redirects While in Queue

A calls HP, call offered on B and B answers (GC1). D calls HP and gets queued (GC2).

Action	Event	Call info
D redirects the call to E	GC2 ConnCreatedEv E GC2 CallCtlCallOfferedEv E GC2 TermConnDroppedEv TermD GC2 CallCtlTermConnDroppedEv termD GC2 ConnDisconnectedEv D GC2 CallCtlTermConnDisconnectedEv D GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E GC2 TermConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE	Ev.getCiscoFeatureReason() = REASON_REDIRECT Call.getCalledAddress() = E Call.getCallingAddress() = HP Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = HP Call.getModifiedCallingAddress() = HP Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = D
E answers	GC2 CallCtlConnEstablishedEv E GC2 CallCtlTermConnTalkingEv termE	
B drops GC1	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlTermConnDisconnectedEv A GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlTermConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertinEv B GC2 CallCtlConnAleringEv B GC2 TermConnCreatedEv Term B Gc2 termConnRingingEv TermB GC2 CallCtlTermConnRingingEv TermB GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = E Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = E Call.getModifiedCallingAddress() = E Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
	GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
B answers	GC2 CallCtlConnEstablishedEv B GC2 CallCtlTermConnTalkingEv termB	

Caller (Observed) Conferences While in Queue

A calls HP, call offered on B and B answers (GC1). D calls HP and gets queued (GC2).

Action	Event	Call info
D initiates a consult call with E	GC2 callCtltermConnHeldEv termD GC3 CallActiveEv GC3 ConnCreatedEv D GC3 CallCtlConnInitiatedEv D GC3 CallCtlConnEstablishedEv D GC3 TermConnCreatedEv D GC3 CallCtlTermConnTalkingEv TermD GC3 ConnCreatedEv E GC3 ConnInProgressEv E GC3 CallCtlConnOfferedEv E GC3 ConnAlertingEv E GC3 CallCtlConnAlertingEv E GC3 CallCtltermConnRingingEv termE	
E answers	GC2 CallCtlConnEstablishedEv E GC2 CallCtlTermConnTalkingEv termE	

Action	Event	Call info
D conferences GC1 with Cg2 GC1.conference(GC2)	GC2 CiscoConfereceStartedEv GC3 TermConnDroppedEv TermE GC3 CallCtlTermConnDroppedEv termE GC3 ConnDisconnectedEv E GC3 CallCtlTermConnDisconnectedEv E GC3 TermConnDroppedEv TermD GC3 CallCtlTermConnDroppedEv termD GC3 ConnDisconnectedEv D GC3 CallCtlTermConnDisconnectedEv D GC3 CallInvalidEv GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP GC2 ConnCreatedEv HP GC2 CallCtlConnEstablishedEv HP GC2 CiscoConferenceEndEv	Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getConnection()).getAddress()).getType() = CiscoAddress.UNKNOWN

Action	Event	Call info
B drops GC1	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEvHP GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertingEv B GC2 CallCtlConnAlertingEv B GC2 CallCtltermConnRingingEv termB Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP GC2 ConnCreatedEv HP Gc2 ConnAlertingEv HP GC2 CallCtlConnAlertingEv HP GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEvHP	Ev.getCiscoFeatureReason() = REASON_NORMAL Ev.getConnection()).getAddress()).getType() = CiscoAddress.UNKNOWN Ev.getCiscoFeatureReason() = REASON_DEQUEUING Ev.getCiscoFeatureReason() = REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT
B answers	Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP GC2 TermConnActiveEv termb GC2 CallCtlTermConnTalkingEv termB	Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Use Cases for NuRD (Number Matching for Remote Destination)

Prerequisites

Pre-conditions to nurd use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Enable "Route calls to all remote destinations" checkbox for CTIRD1 and CTIRD2.
- Clusterwide service parameter "Reroute Remote Destination Calls to Enterprise Number" is set to true.
- ICT Trunk configured with Route pattern is 408XXXXX with no discard digits.
- SIP Trunk configured with Route pattern is 409XXXXX with no discard digits.
- ICT Trunk configured with Route pattern is 33.XXXX with discard digits pre-dot.
- CTIRD1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "RDD1", Number: "40822077")
    - Remote destination 2 (Name: "RDD2", Number: "40922078")
- CTIRD2 associated to user "Mobility2", dn = 9200
    - Remote destination 1 (Name: "RDD3", Number: "40812115")
    - Remote destination 2 (Name: "RDD4", Number: "40912116")
- CTIRD2 has a shared ip phone D
- RDP2 associated to user "Mobility2", dn = 9200 with display and Unicode name configured to be "RDP2".
    - Remote destination 1 (Name: "RDD3", Number: "40812115"). This is shared with CTIRD2.
    - Remote destination 2 (Name: "RDDP1", Number: "40922095")
- Device A (IP Phone - Name: "SEP2401C7824EA3", Line A1 (dn: 9000)).
- Device B (IP Phone - Name: "SEP2401C7824EAE", Line B1 (dn: 9001)).
- User1 has in its control list: Device A, B, CTIRD1 and CTIRD2. All devices and lines are observed.

Basic Calls Initiated From Remote Destination

Table 285: Remote Destination Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDD1 calls A1 in which active rd is not set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConOfferedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Direct call between RDD1 and A.	CallingAddress = 40822077, CalledAddress = 9000, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = External

Table 286: Remote Destination Initiates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1 in which active rd is set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1 so looks like CTIRD1 is initiating a call to A.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = Internal

Table 287: Remote Destination Initiates Call with Active RDD with Only CTIRD Observed

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 observes only CTIRD		
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.	*If active remote destination is not set, then app will not see any call events as it would be direct call between RDD1 and A1.	

Action	Events	Call Info
RDD1 calls A1 in which active rd is set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 *Call is routed through CTIRD1 so looks like CTIRD1 is initiating a call to A.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = Internal

Basic Calls to Remote Destination

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
A1 calls RDD1 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnNetworkReachedEv 40822077 GC1: CallCtlConnNetworkAlertingEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 *Direct call between A and RDD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 40822077 CurrentCalledAddress.getType() = External

Table 288: Call to Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through and offered on CTIRD1 and extended to RDD1 with no delay.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303 CurrentCalledAddress.getType() = Internal

Table 289: Call to Remote Destination with Active RDD with Only CTIRD Observed

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 observes only CTIRD		
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("40822077", true) on CTIRD1.	*If active remote destination is not set, then app will not see any call events as it would be direct call between A1 and RDD1.	

Action	Events	Call Info
A1 calls RDD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through and offered on CTIRD1 and extended to RDD1 with no delay.	CallingAddress = 9000, CalledAddress = 2303, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303 CurrentCalledAddress.getType() = Internal

CTIRD/RDP Interaction

Table 290: Remote Destination Shared Between CTIRD and RDP Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds	ProvInServiceEv	

Action	Events	Call Info
RDD3 calls B1 in which active rd is not set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnInitiatedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through the RDP.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName = RDP2

Table 291: Remote Destination Shared Between CTIRD and RDP Initiates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD3 (dn = 40812115). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40812115", true) on CTIRD2.		

Action	Events	Call Info
RDD3 calls B1 in which active rd is set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: CallCtlConnDialingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through CTIRD2 so looks like CTIRD2 is initiating a call to B.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 292: Remote Destination Unique to CTIRD2 Initates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDD4 calls B1 in which active rd is not set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnCreatedEv 40912116 GC1: ConConnectedEv 40912116 GC1: CallCtlConnEstablishedEv 40912116 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TemConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlTermConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD4 and B1.	CallingAddress = 40912116, CalledAddress = 9001, CurrentCallingAddress = 40912116, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 293: Remote Destination Unique to CTIRD2 Initates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD4 (dn = 40912116). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40912116", true) on CTIRD2.		

Action	Events	Call Info
RDD4 calls B1 in which active rd is set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: CallCtlConnDialingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through CTIRD2 so looks like CTIRD2 is initiating a call to B.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 294: Remote Destination Unique to RDP Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDDP1 calls B1. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnInitiatedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through the RDP.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName = RDP2

Table 295: Call to Remote Destination Shared with RDP with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDD3 in which active rd is not set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40812115"). Call is offered to RDD3 after delay and then answered at RDD3 (dn = 40812115).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 *Call is routed through RDP.	CallingAddress = 9001, CalledAddress = 40812115, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName = RDP2

Table 296: Call to Remote Destination Shared with RDP with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD3 (dn = 40812115). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40812115", true) on CTIRD2.		

Action	Events	Call Info
B1 calls RDD3 in which active rd is set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40812115"). Call is answered at RDD3 (dn = 40812115).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnAlertingEv 9200 GC1: CallCtlConnAlertingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnRingingEv CTIRD2 GC1: CallCtlTermConnRingingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 *Call is routed through and offered on CTIRD2 and extended to RDD3 with no delay.	CallingAddress = 9001, CalledAddress = 40812115, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName =

Table 297: Call to Remote Destination Unique to RDP with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDDP1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40922095"). Call is offered to RDDP1 after delay and then answered at RDDP1 (dn = 40922095).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 *Call is routed through RDP.	CallingAddress = 9001, CalledAddress = 40922095, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName = RDP2

Table 298: Call to Remote Destination Unique to CTIRD with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDD4 in which active rd is not set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40912116"). Call is answered at RDD4 (dn = 40912116).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 40912116 GC1: ConnConnectedEv 40912116 GC1: CallCtlConnNetworkReachedEv 40912116 GC1: CallCtlConnNetworkAlertingEv 40912116 GC1: CallCtlConnEstablishedEv 40912116 *Direct call between B1 and RDD4.	CallingAddress = 9001, CalledAddress = 40912116, CurrentCallingAddress = 9001, CurrentCalledAddress = 40912116 CurrentCalledPartyDisplayName =

Table 299: Call to Remote Destination Unique to CTIRD with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD4 (dn = 40912116). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40912116", true) on CTIRD2.		

Action	Events	Call Info
B1 calls RDD4 in which active rd is set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40912116"). Call is answered at RDD4 (dn = 40912116).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnAlertingEv 9200 GC1: CallCtlConnAlertingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnRingingEv CTIRD2 GC1: CallCtlTermConnRingingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 *Call is routed through and offered on CTIRD2 and extended to RDD4 with no delay.	CallingAddress = 9001, CalledAddress = 40912116, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName =

Multiple Calls

Table 300: Make Multiple Calls From Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000

Action	Events	Call Info
RDD1 calls B1. Call is answered at B1.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnInProgressEv 9001 GC2: CallCtlConnOfferedEv 9001 GC2: ConnCreatedEv 40822077 GC2: ConnConnectedEv 40822077 GC2: CallCtlConnEstablishedEv 40822077 GC2: ConnAlertingEv 9001 GC2: CallCtlConnAlertingEv 9001 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3 GC2: ConnConnectedEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD1 and B1.	CallingAddress = 40822077, CalledAddress = 9001, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9001

Table 301: Make Multiple Calls To Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303

Table 302: Remote Destination First Makes a Call and Then Receives a Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303

Table 303: Remote Destination First Receives a Call and Then Makes a Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
RDD1 calls B1. Call is answered at B1.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnInProgressEv 9001 GC2: CallCtlConnOfferedEv 9001 GC2: ConnCreatedEv 40822077 GC2: ConnConnectedEv 40822077 GC2: CallCtlConnEstablishedEv 40822077 GC2: ConnAlertingEv 9001 GC2: CallCtlConnAlertingEv 9001 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3 GC2: ConnConnectedEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD1 and B1.	CallingAddress = 40822077, CalledAddress = 9001, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9001

Table 304: Persistent Call Exists and Remote Destination Initiates Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
Create persistent call on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Direct call between RDD1 and A1	CallingAddress = 40822077, CalledAddress = 9000, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9000
Disconnect the persistent call. Call disconnects successfully.		
A drops the call. Call disconnects successfully.		

Table 305: Persistent Call Exists and Make Call to Remote Destination

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
Create persistent call on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is offered on CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303
Disconnect the persistent call. Disconnect throws exception.		Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_CALL_ACTIVE
A drops the call. Call disconnects successfully.		
Disconnect the persistent call. Call disconnects successfully.		

Table 306: Call to Remote Destination with Active RDD and Call Forward All Configured on CTIRD with Only CTIRD Observed

Action	Events	Call Info
Configure CallForwardAll on CTIRD1 to 2302.		
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Nurd features disables CFA.	CallingAddress = 9000, CalledAddress = 2303, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Table 307: Max Calls Limit Reached Where CTIRD Has 2 Calls to the Remote Destination and CTIRD Still Tries to Make a Call

Action	Events	Call Info
Set Max Calls = 2 and Busy Trigger = 2		
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303
CTIRD makes outgoing call to 2302. User1 invokes Call.connect ("CTIRD1", "2303", "2302"). Fail to make outgoing call since max calls limit reached so throws exception.		Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_MAXCALL_ LIMIT_REACHED
Revert back to orig values. Set Max Calls = 4 and Busy Trigger = 2		

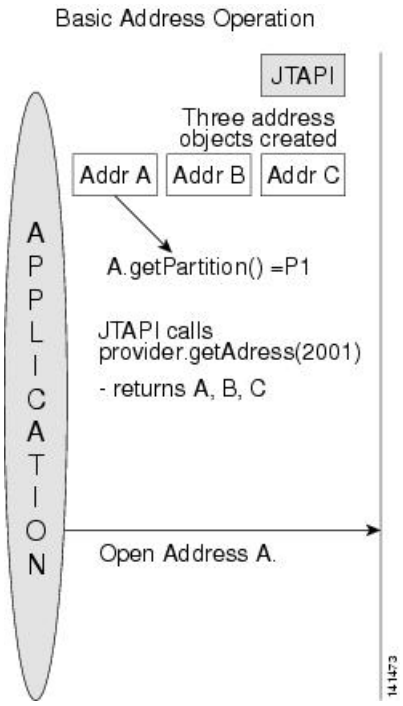
Partition Support

Since the address hashing mechanism in JTAPI has changed, this feature is expected to have performance degradation in address lookup time and during load tests.

Using getPartition() API

The example given below illustrates how getPartition(), will be used by JTAPI and applications, to differentiate between addresses having same DN but belonging to different partitions.

Using getPartition() API



In this case, there are three addresses which belong to three different partitions: A(2001, P1), B(2001, P2) and C(2001, P3), where 2001 indicates DN and P1, P2, and P3 denote different partitions.

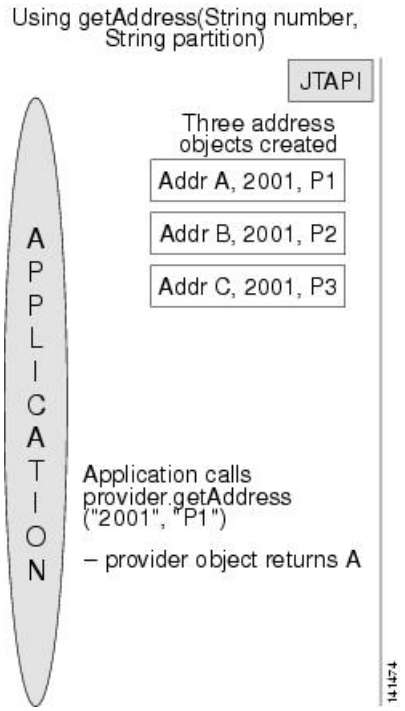
When JTAPI calls provider.getAddress("2001"), the provider object will return an array of three address objects containing A, B and C, since all of them have the same DN.

The application and JTAPI will distinguish between the three addresses by using the getPartition() method of the address object.

Using getAddress (String Number String Partition)

Consider the example shown below to see how JTAPI will use the getAddress (String number, String partition) API to retrieve the address object corresponding to a particular DN and partition when there are multiple addresses with same DN and different partitions.

Using getAddress (String Number, String Partition)



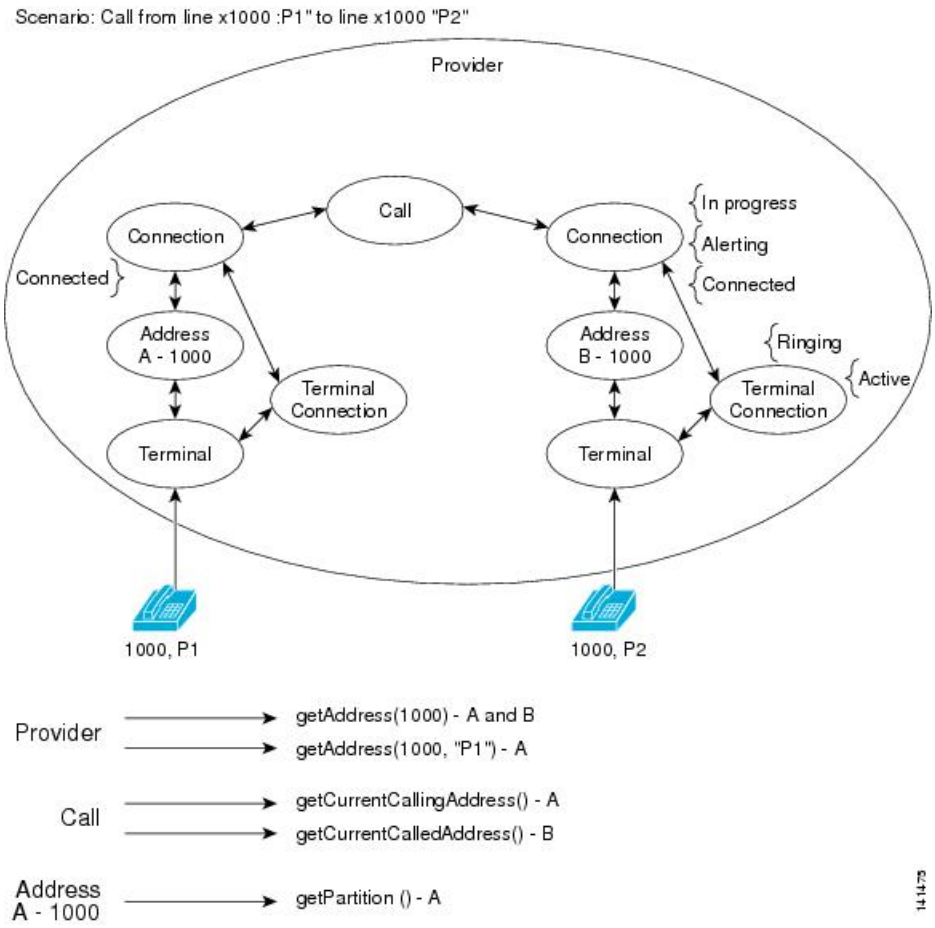
In this case, there are three addresses which belong to three different partitions. Let us denote them by A(2001, P1), B(2001, P2) and C(2001, P3), where 2001, indicates DN and P1, P2, P3 denote different partitions.

When JTAPI calls provider.getAddress(“2001”, “P1”)**,** the provider object will return the address object which has the same DN i.e. 2001 and the same partition info, that is P1–as provided in the API. In this case, the address object A will be returned to the application.

Simple Call Scenario

Consider the following scenario where A calls B. A has DN 1000 and calls B which also has DN 1000. A belongs to partition P1 and B belongs to partition P2. The following diagram illustrates the various events and the results of API calls pertaining to this scenario, which are relevant to partition support feature.

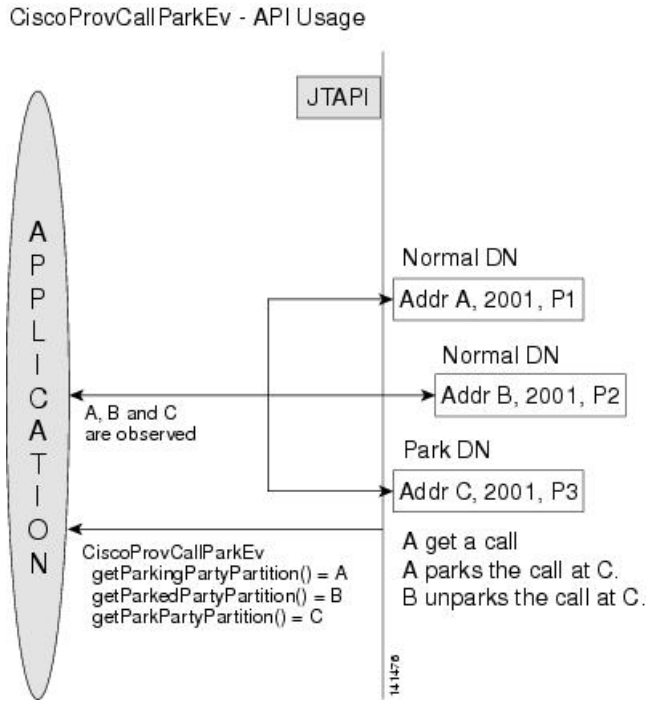
Figure 20: Simple Call Scenario



Park DN

Park DNs are also treated as addresses in JTAPI. Hence, the same treatment given to normal DN is also given to park DN. The following message flow illustrates how an application will use park DN partition information in a call where park DNs are involved.

Park DN Scenario



When the application is monitoring park DN, it is possible to have the park DN to be the same as a regular DN (while both belong to different partitions).

In this case, C is a park DN having same DN value as A and B while belonging to a different partition.

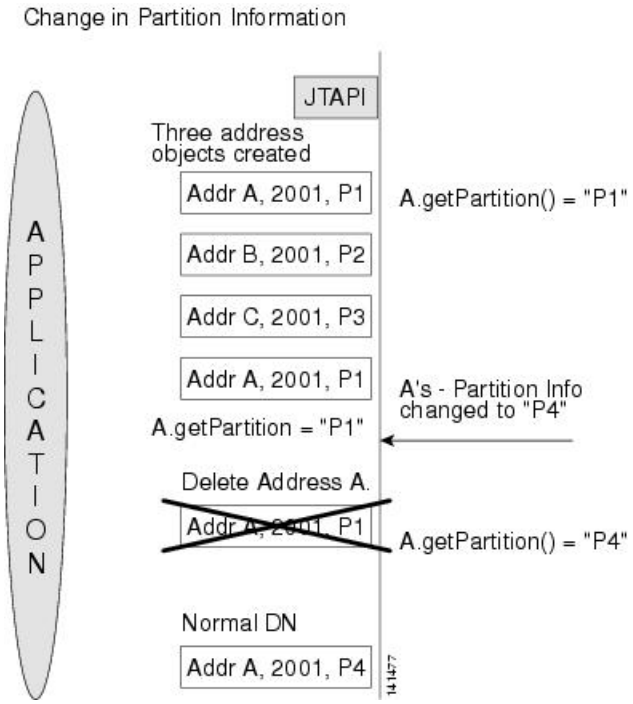
A receives a call and parks the call at C. B unparks the call. While the call is parked, and unparked, CiscoProvCallParkEv is generated. The API

getParkingPartyPartition(), getParkedPartyPartition() and getParkPartyPartition() return the associated address objects as shown in the figure.

Partition Change

Partition attribute is similar to the DN attribute of an address. Hence, whenever the partition attribute changes, the address object has to be destroyed and recreated. When the partition information of an address is changed, JTAPI will be restarted during which the current address objects will be deleted and new address objects will be created, reflecting the changed partition information.

Change in Partition



When the partition information of an address is changed, the address object will be destroyed and a new address object will be created.

The new address object will have the new partition information.

In the example given, Address A's partition string was changed to P4. Hence, the current address object of A will be deleted and a new address object will be created.

A query on the old address object using A.getPartition() will retrieve “P1”, while the same query on the new object will return “P4”.

When the address partition changes, applications should query the address objects to update their partition information.

JTAPI Partition Support

The common assumption for all of the following use cases is that CTI provides partition information for all the lines which the JTAPI opens in the response message and JTAPI stores the partition information for every address it maintains.

S.No.	Pre-Condition	Scenario	Expected Behavior	Result
1	A and B are two addresses in the same cluster with same DN and different partitions (P1, P2) and in same cluster. A calls B. CSS of A has B's partition in it first.	Calls between addresses with same DN in different device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B
2	A and B are two addresses with same DN in same device but different partitions (P1, P2) and in same cluster. A calls B. CSSof A has B's partition in it first.	Calls between addresses with same DN in same device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B.
3	A, B, and C are three different addresses with different DN. (P1, P2, P3). A park DN is configured with same DN as C and different partition (P4).	A calls B. B parks the call. C unparks the call from a park DN which is same as C's DN.	JTAPI should allow C to unpark the call from park DN.	C is successfully able to unpark the call from park DN.
4	A, B, and C are three different addresses having the same DN and different partitions (P1, P2, P3). Apark DN is configured with same DN but belonging to a different partition (P4)	A calls B, B parks call at park DN. C unparks the call.	JTAPI should allow C to unpark the call from park DN.	A is able to call B. B should be able to park the call at the park DN. C should be able to pick up the call.
5	A, B, and C are three different addresses with same DN and different partitions (P1, P2, and P3)	JTAPI calls getPartitionAddress(DN of A).	Three address objects are returned each corresponding to A, B and C.	JTAPI maintains the address objects based on partition info and DN.
6	A and B are addresses with same DN but belong to different partitions (P1, P2).	Application calls getPartition() on the address objects of A and B.		Partition strings of the addresses are returned correctly.
7	A and B are two different addresses (different DNs) belonging to different partitions. A and B are in the control list of the same user. Provider open is completed and the lines are opened.	JTAPI supports old API to open lines when DN is different. There will be no change in behavior.	Lines A and B will be opened, but since they have different DN, user need not specify the partition info. DN alone is sufficient to open the lines, but users can also give partition info and both modes of opening lines will be supported by JTAPI.	Address objects for A and B are created successfully.

S.No.	Pre-Condition	Scenario	Expected Behavior	Result
8	A is an address in the control list of user and is opened by the user. From the CM admin page the partition information of A is changed. The device is restarted after this.	Partition information of a DN is changed. JTAPI should reflect new partition information.	JTAPI will delete the current address object and create a new address object for A when it comes in service again. By querying the partition info of this address object, user should get changed partition info.	New partition information is reflected in the new address object.

Persistent Connection Use Cases

The following pre-conditions apply to all persistent call use cases, unless specified:

- The provider is in IN_SERVICE state.
- All addresses and terminals are already in service.
- Device A (CTI Remote Device - Name: "CTIRDtapi", Line A1 (dn: 881000))

  Remote destination 1 (Name: "rd", Number: "78000")
- Device B (IP Phone - Name: "SEP001319ACCA26", Line B1 (dn: 1000))
- Device C (IP Phone - Name: "SEP00156247EE60", Line C1 (dn: 2000))
- User1 has in its control list: Devices A, B and C. All devices and lines are observed.

Table 308: Call createPersistentCall() on an Address That Is Not Configured to a Remote Terminal Device, i.e. on an IP Phone

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall ("SEP00156247EE60", "5000", "remote") on device C.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED _ON_DEVICE.

Table 309: Call createPersistentCall()on an Address That Is Configured to a Remote Terminal Device Where Active RD Is Not Set

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	Caught exception com.cisco.jtapi.PlatformException: The active remote destination is not set.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_REMOTE_DEVICE_REQUEST _FAILED_ACTIVE_RD_NOT_SET

Table 310: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD Is Set; Verify That Persistent Call Is Connected

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal .setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Action	Events	Call Info
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
User1 invokes CiscoAddress. getPersistentConnection ("CTIRDjtapi") and verify that the connection for the persistent call is returned and uses that to get the Call object and confirm it is for the persistent call.		((CiscoAddress.getPersistentConnection ("CTIRDjtapi")).getCall()).isPersistentCall() = true.
User1 invokes Provider.getCalls()		Provider.getCalls() = null
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A = null
User1 invokes Terminal. getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A = null
Disconnect/drop the persistent call. User1 invokes either Call.drop() or Connection.disconnect()	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 311: Call createPersistentCall() on an Address Configured to a Remote Terminal Device Where a Persistent Call Already Exists

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "6000", "remote2") on device A.	Caught exception com.cisco.jtapi.PlatformException: Persistent Call exists.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_PERSISTENT_CALL_EXISTS.

Table 312: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active Rd Is Set; Verify That Persistent Call Is Connected and Then Have Remote Destination Hang Up

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Remote destination with dn = 78000 hangs up.	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 313: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True; Verify That Persistent Call Is Connected; Set Active RD = False and Verify That Persistent Call Is Dropped

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", false) on device A.	CiscoProvTerminalRemote DestinationChangedEv See persistent call gets dropped: GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = false

Table 314: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True; Verify That Persistent Call Is Connected; Make Incoming Customer Call to Same Remote Terminal Device

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.

Actions	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Call.connect("SEP001319ACCA26", "1000", "8881000")	GC2: CallActiveEv GC2: ConnCreatedEv 1000 GC2: ConnConnectedEv 1000 GC2: CallCtlConnInitiatedEv 1000 GC2: TermConnCreatedEv SEP001319ACCA26 GC2: TermConnActiveEv SEP001319ACCA26 GC2: CallCtlTermConnTalkingEv SEP001319ACCA26 GC2: CallCtlConnDialingEv 1000 GC2: CallCtlConnEstablishedEv 1000 GC2: ConnCreatedEv 8881000 GC2: ConnInProgressEv 8881000 GC2: CallCtlConnOfferedEv 8881000 GC2: ConnAlertingEv 8881000 GC2: CallCtlConnAlertingEv 8881000 GC2: TermConnCreatedEv CTIRDjtapi GC2: TermConnRingingEv CTIRDjtapi GC2: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 1000, CalledAddress = 8881000, CurrentCallingAddress = 1000, CurrentCalledAddress = 8881000
Call is answered at device A	GC2: ConnConnectedEv 8881000 GC2: CallCtlConnEstablishedEv 8881000 GC2: TermConnActiveEv CTIRDjtapi GC2: CallCtlTermConnTalkingEv CTIRDjtapi	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", false) on device A.	CiscoProvTerminalRemote DestinationChangedEv Both persistent call with GC1 and customer call with GC2 are not dropped/disconnected even though active rd = false.	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = false.

Actions	Events	Call Info
Customer call with GC2 is disconnected/dropped. User1 invokes either Call.drop() or Connection.disconnect() on the call with GC2.	GC2: TermConnDroppedEv SEP001319ACCA26 GC2: CallCtlTermConnDroppedEv SEP001319ACCA26 GC2: ConnDisconnectedEv 1000 GC2: CallCtlConnDisconnectedEv 1000 GC2: TermConnDroppedEv CTIRDjtapi GC2: CallCtlTermConnDroppedEv CTIRDjtapi GC2: ConnDisconnectedEv 8881000 GC2: CallCtlConnDisconnectedEv 8881000 GC2: CallInvalidEv Since there are no active calls on device A and active rd is now false, the persistent call with GC1 is now dropped/disconnected. GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 315: Have a Persistent Call and Customer Call Connected; Invoke hold() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		

Actions	Events	Call Info
Invoke hold() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON _PERSISTENT _CALL.

Table 316: Have a Persistent Call and Customer Call Connected; Invoke startRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke startRecording() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON_ PERSISTENT _CALL.

Table 317: Have a Persistent Call and Customer Call Connected; Invoke stopRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke stopRecording() on the persistent call with GC1. Make sure Selective call recording is enabled.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON _PERSISTENT _CALL.

Table 318: Have a Persistent Call and Customer Call Connected; Invoke conference() on the Persistent Call Where Persistent Call Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Actions	Events	Call Info
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where persistent call with GC1 is the primary call and customer call with GC2 is the secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 319: Have a Persistent Call and Customer Call Connected; Invoke conference() on the Persistent Call Where Persistent Call Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where customer call with GC2 is primary call and persistent call with GC1 is secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 320: Have a Persistent Call and Customer Call Connected; Invoke park() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke park().	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 321: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(Call) where persistent call with GC1 is primary call and customer call with GC2 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 322: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Primary to Another DN Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(String address) where persistent call with GC1 is primary call to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 323: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		

Actions	Events	Call Info
Invoke transfer(Call) where customer call with GC2 is primary call and persistent call with GC1 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 324: Have a Persistent Call and Customer Call Connected; Invoke consult() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Make consult call from device A to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 325: Have a Persistent Call and Customer Call Connected; Invoke pickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke pickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 326: Have a Persistent Call and Customer Call Connected; Invoke otherPickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke otherPickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 327: Have a Persistent Call and Customer Call Connected; Invoke redirect() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke redirect("2000") on the persistent call.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Play Announcement

Prerequisites

Pre-conditions to all play announcement use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Device A (CTI Remote Device - Name: "CTIRD3", Line A1 (dn: 9202))
    - o Remote destination 1 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- Device B (IP Phone - Name: "SEP2401C7824EA3", Line B1 (dn: 9000))

- Announcement Identifier is Welcome Greeting Sample.
- User1 has in its control list: Devices A, and B. All devices and lines are observed.

Basic Play Announcement Use Cases

Basic Play Announcement Use Cases

Table 328: Play Announcement on CTI Remote Device with Persistent Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnOfferedEv Unknown	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
User1 invokes Provider.getCalls()		Provider.getCalls() returns the announcement call.
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for the announcement call.

Table 329: Play Announcement That Stopped Playing Before the End of the Announcement

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call info
User1 invokes Provider.getCalls()		Provider.getCalls() returns the announcement call.
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for the announcement call.
Disconnect/drop the announcement call. User1 invokes either Call.drop() or Connection.disconnect() to stop the announcement before it finishes playing.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 330: Play Announcement with Incoming Customer Call in Ringing State

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
User1 invokes Provider.getCalls()		Provider.getCalls() returns both the customer call and the announcement call
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for both the customer call and the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for both the customer call and the announcement call.

Action	Events	Call Info
After announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 331: Play Announcement Where the Call Is Answered Before the Full Message Plays

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	

Action	Event	Call Info
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
Customer call is answered. Announcement call is dropped/disconnected.	GC3: CallCtlTermConnHeldEv CTIRD3 GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 332: Play Announcement Where the Call Is Answered Before the Full Message Plays

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	

Action	Event	Call Info
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
Customer call is answered. Announcement call is dropped/disconnected.	GC3: CallCtlTermConnHeldEv CTIRD3 GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 333: Play Announcement with Call in Connected State

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not available in current state.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. OPERATION_NOT_AVAILABLE_ IN_CURRENT_STATE.

Table 334: Play Announcement with Held Customer Call

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	
User1 puts the customer call on hold. Invokes TerminalConnection.hold() on Device A.	GC2: CallCtlTermConnHeldEv CTIRD3	

Action	Event	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
After the announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 335: Play Announcment with an Outgoing Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
Device A makes outgoing call to Device C. User1 invokes Call.connect("CTIRD3", "9202", "9000"). Call is left unanswered (ringing state).	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: ConnConnectedEv 9202 GC2: CallCtlConnInitiatedEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: CallCtlConnDialingEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv 9000 GC2: ConnInProgressEv 9000 GC2: CallCtlConnOfferedEv 9000 GC2: ConnAlertingEv 9000 GC2: CallCtlConnAlertingEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3	

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: TermConnDroppedEv SEP2401C7824EA3 GC2: CallCtlTermConnDroppedEv SEP2401C7824EA3 GC2: ConnDisconnectedEv 9000 GC2: CallCtlConnDisconnectedEv 9000 GC2: CallInvalidEv GC2: CallObservationEndedEv GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call Info
After announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 336: Play Announcement on an IP Phone

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("SEP8478ACE7F9B9", "Welcome Greeting Sample") on ip phone.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED_ ON_DEVICE.

Table 337: Play Announcement on the CTI Port

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("CTI3", "Welcome Greeting Sample") on CTI Port.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED_ ON_DEVICE.

Table 338: Play Announcement on CTI Remote Device Without Persistent Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: No persistent call exists.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_NO_PERSISTENT_ CALL_EXISTS.

Table 339: Play Announcement on a CTI Remote Device While a Persistent Call Is Being Set Up

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
Call is offered to the remote destination but not picked up.		
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Persistent Call is being set up.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_PERSISTENT_CALL_ BEING_SETUP.

Table 340: Play Announcement on CTI Remote Device with a Persistent Call with an Invalid Announcement Identifier

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Invalid Media resource ID.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_MEDIA_ RESOURCE_ID.

Table 341: Play Announcement Back to Back

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	For these 3 call events, feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call Info
After announcement finishes playing, the call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Announcement already in progress.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_ANNOUNCEMENT_ALREADY_IN_PROGRESS.

Table 342: Play Announcement to Stop IPVMS Service

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
Stop IPVMS service on all the nodes.		

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown GC2: CiscoAnnouncementErrorEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementErrorEv. getErrorCode() = -1932787536 CiscoAnnouncementErrorEv. getErrorDescription() = Resource Not Available.

Play Announcement Feature Interaction Use Cases

Play Announcement Feature Interaction Use Cases

Table 343: Hold Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes TerminalConnection.hold () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Table 344: Redirect Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes Connection.redirect () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Table 345: Park Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes Connection.park () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Play Zip Tone

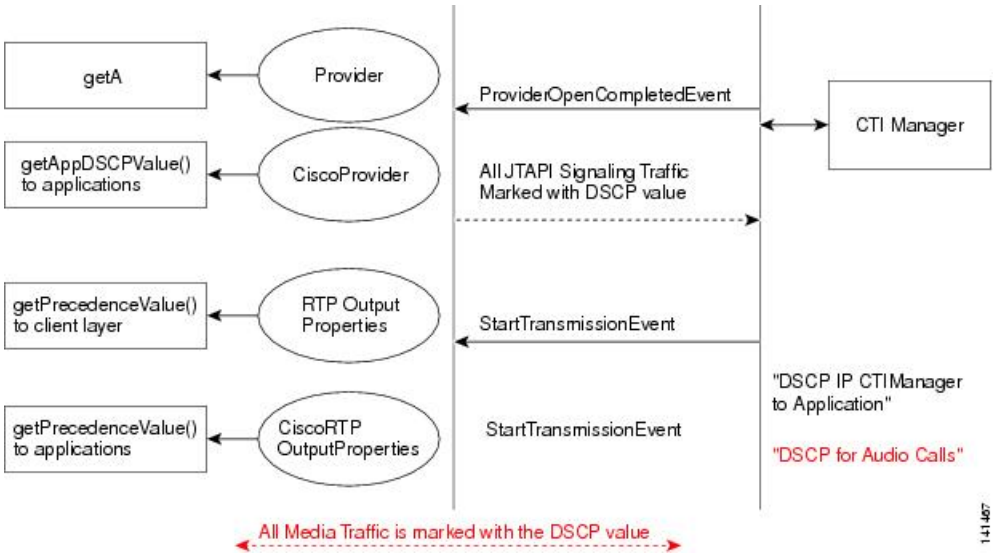
A and B represents the terminals. TermConnB represents the Cisco terminal connection of Terminal B. TermConnCTI1 represents the Cisco terminal connection of terminal CTI1. CTI1 is a Cisco media terminal.

Sl.No	Scenario	Events/Response
1.	A calls B. B answers the call and the application invokes TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	User on B hears the ZIP tone.
2.	A calls B. B answers the call and application invokes TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	User on A hears the ZIP tone.
3.	A calls B. B is alerting. Application calls TermConnB. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	Tone is heard by user B.
4.	A calls CTI1. Call is answered by CTI1. Application calls TermCTI. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	PlatformException is thrown to application request.

Sl.No	Scenario	Events/Response
5.	A calls CTI1. Call is answered by CTI1. Application calls TermConnCTI1. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	User on A hears the ZIP tone.
6.	A, B and CTI1 are in conference. Application calls TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	User on B hears the ZIP tone.
7.	A, B and CTI1 are in conference. Application calls TermB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	None of the users on A, B and CTI hear the tone.

QoS Support

Figure 21: Call Flow Diagram for QoS Support



JTAPI QoS

For QoS to work in Windows, complete the following steps:

1. Start Registry Editor (Regedt32.exe).
2. Go to the following key:

HKEY_LOCAL_ MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\

Note The registry key is one path.

1. On the Edit menu, click Add Value.
2. Type DisableUserTOSSetting.
3. Click REG_DWORD in the Data Type box.
4. Click OK.
5. Enter 0 in the prompt box.
6. Quit the Registry Editor.
7. Restart the computer.

For more information on this see http://support.microsoft.com/default.aspx?scid = kb;en-us;248611

Scenario	JTAPI Behavior
Application uses the JTAPI getPrecedenceValue() API to query for the new DSCP value, in CiscoRTPOutputStartedEvent.	JTAPI returns the DSCP value received from CTI in StartTransmissionEvent to the application.
Application uses the JTAPI getAppDSCPValue() API to query for the new DSCP value, when it gets ProviderInServiceEvent.	JTAPI returns the DSCP value received from CTI in ProviderOpenCompletedEvent to the application.

QSIG Path Replacement

The following table shows the JTAPI events that are delivered to applications when calls between PBXs that are connected by Q.Signaling (QSIG) trunks are transferred and forwarded. This table also shows the events that are delivered to applications when the real-time path (RTP) is optimized by the QSIG Path Replacement feature.

Calls going out on a QSIG trunk may not have a connection for the far end if any translation pattern is changing the pattern. In other words, when the application sees two calls in the trombone case, B may not serve as the common connection on the calls.

No.	Action	Event
1	A registered with CM1, B is registered with CM2, and C registered with CM3. A calls B (GC1); B transfers the call to C. The application is monitors C. The PR feature replaces the path after the call gets connected to C. The same action applies to scenarios that involve call forward at B. (The called party transfers the call.)	These events get delivered to applications: CallCtrlConnectionEstablishedEv A CallCtrlConnectionDisConnectEv B OpenLogicalChannelEvent if C is a CTI device (Dynamically registered CTIPorts and RP)

No.	Action	Event
2	A registered with CM1, B registered with CM2, and C registered with CM3. B calls C; C answers; B transfers the call to A. A answers. The application is monitors only C. (The calling party transfers the call.)	In this case, both A are C represent called parties when transfer completes. After the call is answered, PR replaces the path. In this case, A and C represent IP phones; the display will be updated as a part of PR feature operation, that makes either A or C as calling. JTAPI events: GC1: CallCtlConnEstablishedEv A GC1: CallCtlConnDisconnectedEv B
3	Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. A calls B (GC1); B answers and transfers the call to C (GC2). Path replacement connects A and C bypassing CM2. The application observes both A and C. (The called party transfers the call.)	For GC1 Call Observer: GC1: CallCtlConnEstablishedEv C GC1: CallCtlConnDisconnected B Before the PR feature replaces the path, the application sees two calls: GC1 with connections to A and C (external) and GC2 with connections to C and A (external). When the PR feature replaces the path, either GC1 changes GC2, or GC2 changes to GC1. Assuming A's GCID changes from GC1 to GC2: GC1: CiscoCallChangedEv (oldGCID = GC1, newGCID = GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: TermConnTalkingEvent for TerminalA cause = CAUSE_QSIG_PR

No.	Action	Event
4	Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. B calls A and transfers the call to C. Path replacement connects A and C, bypassing CM2. Application observes both A and C. (The calling party transfers the call.)	Before the PR feature replaces the path, the application will see two calls: GC1 with connections to A and B (external) and GC2 with connections to C and B (external). In this case, the application will not see any transfer start events. When PR feature replaces the path, it updates the display of A and C and path gets replaced, resulting in a GCID change. Assuming A's GCID is changed and made the calling party, the following JTAPI events occur: GC1: CiscoCallChangedEv (GC1 to GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: ConnCreatedEv A GC2: ConnConnectedEv A GC2: TermConnTalkingEvent for TerminalA cause = CAUSE_QSIG_PR
5	A registered to CM1, B registered to CM2, and C registered to CM1. A calls B; B transfers the call to C. C answers and before path replacement completes, C invokes a feature (park, redirect, and so on).	Path replacement gets abandoned.
6	In some conditions, call processing ignores feature requests (redirect, park, transfer, and so on). This happens when a setup request is sent out, and the local CM is waiting for connect from the other end.	JTAPI: Exception will be thrown from JTAPI for feature requests.
7	In some cases, the application could receive dead air when CM goes down when the PR feature is trying to switch the RTP path. This could happen to a previously connected call.	No events JTAPI Apps: Hang up the call

Recording Use Cases

Expose ClusterID in CiscoProvider Interface

Actiions	Events	Call Info
Start application and initialize JTAPI		
Add provider observer	ProvInServiceEv	
Application calls ciscoProvider.getClusterID()		JTAPI returns 'StandAloneCluster'

Admin changes the clusterID to '3NodeClusterinSanJose' and restarts the services		
	ProvOutofService ProvInServiceEv	Application calls getClusterID() - JTAPI returns ' 3NodeClusterinSanJose'

Multi-Clusters Gateway Recording Use Cases

Test configurations Specification:

- Cluster 1 = SME Cluster with 1 PUB and 1 SUB (Note: Only use DN's 2xxx, 3xxx, 9xxx)
- Cluster 2 = External Cluster (Note: Only use DN's 1xxx, 4xxx)
- Cluster 3 = Leaf Cluster (Note: Only use DN's 5xxx, 6xxx)

Assumption: All devices have BIB enabled unless specified in the detailed test cases. CTIRD does not support BIB.

- Cluster 1 and Cluster 2 are connected thru SIP GW which is recording enabled.
- Cluster 1 and Cluster 3 are connected thru SIP trunk ICT.
- Cluster 2 and Cluster 3 are connected thru SIP trunk ICT.

On Cluster 1 Route-Patterns:

- 180.XXXX: SIP ICT trunk from cluster 1 to cluster 3, calledPartyTranformation: remove PreDot.
- 171.XXXX: SIP GW from cluster 1 to cluster 2, calledPartyTranformation: remove PreDot.

On Cluster 2 Route-Patterns:

- 172.XXXX: SIP GW from cluster 2 to cluster 1, calledPartyTranformation: remove PreDot.
- 172180.XXXX: SIP GW from cluster 2 to cluster 1 to cluster 3, calledPartyTranformation: remove PreDot.
- 180.XXXX: SIP ICT trunk from cluster 2 to cluster 3 directly, calledPartyTranformation: remove PreDot.

On Cluster 3 Route-Patterns:

- 172.XXXX: SIP ICT trunk from cluster 3 to cluster 1, calledPartyTranformation: remove PreDot.
- 172171.XXXX: SIP GW from cluster 3 to cluster 1 to cluster 2, calledPartyTranformation: remove PreDot.
- 171.XXXX: SIP ICT trunk from cluster 3 to cluster 2 directly, calledPartyTranformation: remove PreDot.

Recording IP Phones

Scenario 1

IP phones (Basic): Multi-Clusters Gateway preferred with auto recording

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
• Phone B (SEP8E0E) has line B1:4006
  • GW preferred
  • auto recording enabled
  • recorder B:2000
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 3601 configured as:
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. Verify A1 have recording type AUTO_RECORDING
4. A1 (cluster3) call B1 (cluster2) (e.g. 3602 call 1721714006)
5. B1 answers the call
6. A1 drops
	Step 3- Check A1 and B1 CiscoAddres.AUTO_RECORDING
	Step 5- Check two recording sessions established on A1 and B1.
	Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name – SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = recorder B device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE
	.getMediaForkingDeviceName() = device name of B
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster2
	Step 6- Check A1 and B1 get CiscoTermConnRecordingEndEv and disconnect on recorders.

Scenario 2

IP phones (Basic): Multi-Clusters Gateway preferred with auto recording

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP8E0E) has line B1:4006 • GW preferred • auto recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEP3B5F) has line A1 3601 configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** Verify A1 have recording type SELECTIVE_RECORDING **4.** A1 calls B1 (3601 calls 1721714006) **5.** B1 answers **6.** A1 requests startRecording(app) **7.** A1 requests stopRecording() **8.** A1 disconnects B1	Step 3- Check A1 CiscoAddres.SELECTIVE_RECORDING Step 5- Check auto recording started on B1. Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check B1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = recorder B device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of B .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster2 Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check A1 get CiscoTermConnRecordingEndEv and disconnect on recorders.

Scenario 3

IP phones (Basic): Multi-Clusters Gateway preferred with press key invoke recording

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP723F) has line B1:4008 • GW preferred • selective recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEPDB17) has line A1 2303 configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** Verify A1 have recording type SELECTIVE_RECORDING **4.** A1 calls B1 (2303 calls 1721714008) **5.** B1 answers **6.** Press recording key on A **7.** A1 disconnects B1	Step 3- Check A1 and B1 CiscoAddres.SELECTIVE_RECORDING Step 6-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_DEVICE .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1

Scenario 4

IP phones (Basic): Multi-Clusters Gateway preferred with selective recording and Cluster 1 fork media to branch recorder on cluster3

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
 	
Cluster 2:
• Phone B (SEP723F) has line B1:4008
  • GW preferred
  • selective recording enabled
  • recorder B:2000
 	
Cluster 3:
• Branch recorder: 2000
• Phone A (SEPDB17) has line A1 1623 configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile2: 1802000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (1623 call 1721714008)
4. B1 answer the call
5. Start silent recording on A1
6. Stop recording on A1
7. A1 drop
	Step 5- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A2 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = SIPICT-To-Cluster3
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk name – SIP GW.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Check A1 get CiscoTermConnRecordingEndEv

Scenario 5

IP phones (Basic): Multi-Clusters Device preferred with selective recording and device fork media to central recorder

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
• Phone B (SEP723F) has line B1:4008
  • selective recording enabled
Cluster 3:
• Branch recorder: 2000
• Phone A (SEPDB17) has line A1 2008 configured as:
  • Device preferred
  • selective recording enabled
  • recording profile:rec_profile2: 1722000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (A1 call 70662050
4. B1 answer the call
5. A1 request startRecording(app)
6. A1 request stopRecording()
7. A1 drop
	Step 5-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 1722000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE
	.getMediaForkingDeviceName() = device name of A
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster3
	Step 6- Check A1 get CiscoTermConnRecordingEndEv and disconnect on recorder.

Scenario 6

IP phones (Basic): Multi-Clusters Hold and resume - Gateway preferred with automatic recording

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
• Phone B (SEP8E0E) has line B1:4006
  • GW preferred
  • auto recording enabled
  • recorder B:2000
Cluster 3:
• Branch recorder: 2000
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile1: 2000
• Phone A (SEP3B5F) has line A1 3602 configured as:
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (3602 call 1721714006)
4. B1 answer the call
5. A1 put call on hold
6. A1 resume the call
7. A1 drop
	Step 4-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name – SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = recorder B device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk 2 device name – SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster2
	Step 5- Check A1 get CiscoTermConnRecordingEndEv
	Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Step 7- Check A1 get CiscoTermConnRecordingEndEv

Scenario 7

IP phones (Basic): Hold and resume - Phone preferred with Selective Recording

Phone A has line : 1506 Phone B has line : 1504 • Phone Prefered • Selective Recording enabled On Phone B • Recorder Phone C : 1505 • Recorder Phone D : 1503 • recording profile: rec_profile : 1505 **1.** Open provider **2.** Observe A, B, C, D in Prov1 **3.** A call B (1506 calls 1504) **4.** B answer the call. **5.** Selective recording was initiated on Phone B. **6.** B put call on hold. **7.** B resumes the call. **8.** A drop the call.	Step 4-Check B get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check B TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN = 1505) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = Phone Name .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster Check B TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = recorder B device name .getAddress() = Recording profile (DN 1505) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = Phone name .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster Step 6- Check B get CiscoTermConnRecordingEndEv Step 7- Check B get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv Step 8- Check B get CiscoTermConnRecordingEndEv

Scenario 8

IP phones (Basic): Multi-Clusters Multiple calls - Gateway preferred with automatic recording

Setup and Action	Events and Call Info

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP8E0E) has line B1 1681 • GW preferred • selective recording enabled • recorder B:2000 • Phone C (SEP723F) has line B1: 4008 • GW preferred • selective recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEPDB17) has line A1 3602 configured as: • GW preferred • auto recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1, C1 in Prov2 **3.** A1 call B1 (3602 call 1721714006) **4.** B1 answer the call **5.** A1 put call on hold **6.** A1 call C1 (3602 call 1721714008) **7.** C1 answer **8.** A1 retrieve first call **9.** A1 retrieve second call **10.** A1 drop second call	Step 4- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recording device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 5- Check A1 get CiscoTermConnRecordingEndEv Step 7-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. (Recording is started on A1 for call A1->C1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 8- Retrieve first call. Check A1 get CiscoTermConnRecordingEndEv (for A1->C1) Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv (for A1->B1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() =

Setup and Action	Events and Call Info
11. A1 drop first call	.getMediaForkingClusterID() = name of cluster1 Step 9- Retrieve second call. Check A1 get CiscoTermConnRecordingEndEv (for A1->B1) Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv (for A1->C1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1

CTI Remote Devices Use Cases

Scenario 9

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with automatic recording- IP phone (cluster3) call remote device (cluster1)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
• devB (CTIRD3) has line B1:9008 (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
 	
Cluster 2:
• Phone C has line C1 (1681)
 	
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile2: 2000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 (cluster 3) call B1 (cluster1) (3601 call 1729008)
4. Remote destination on cluster 2 answer the call
5. A1 drop
	Step 4- Check auto recording started on A1.
	Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check A1 get CiscoTermConnRecordingEndEv
	Note: Auto recording start on B1 (new changes)
	Step 4-
	Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check B1 get CiscoTermConnRecordingEndEv

Scenario 10

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with silent recording- IP phone (cluster3) call remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
Cluster 2:
• devB (CTI_RD3) has line B1:1625 (remote destination: 1722602 on cluster 1) configured as:
  • selective recording enabled
  • GW preferred
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (3601 calls 1721711620)
4. Remote destination on cluster 1 answer the call
5. Start silent recording on A1
6. Start silent recording on B1
7. Stop recording on A1
8. A1 drop
	Step 5- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check A1 get CiscoTermConnRecordingEndEv
	Note: Auto recording start on B1 (new changes)
	Step 6-
	Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check B1 get CiscoTermConnRecordingEndEv

Scenario 11

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with automatic recording- Remote device (cluster3) call Remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
Cluster 2:
• devB (remote device CTI_RD2) has line B1:1624 (remote destination: 2303 on cluster 1) configured as:
  • selective recording enabled
  • GW preferred
Cluster 3:
• Branch recorder: 2000
• devA (remote device CTI_RD3) has line A1 1622 (remote destination: 9000 on cluster 1) configured as:
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile2: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 (cluster 3) call B1 (cluster2)
4. Call is offered to devA's remote destination
5. Remote destination of devA answer
6. Call is offered to B1
7. DevB's remote destination answer the call
8. Remote destination of devB drop (or devB drop)
	Step 7- Check auto recording started on A1.
	Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1

Scenario 12

CTI Remote Devices (Basic): Multi-Clusters Hold and resume with automatic recording-IP phone call (cluster3) remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 • Mobile through PSTN GW in Cluster 1 Cluster 2: • devB (CTI_RD3) has line B1:1620 (remote destination: 1722602 on cluster 1) configured as: • auto recording enabled • GW preferred Cluster 3: • Branch recorder: 2000 • devA (IP phone) has line 3602 configured as: • GW preferred • auto recording enabled • recording profile:rec_profile2: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** A1 (cluster 3) call B1 (cluster2) **4.** Remote destination of devB answer the call **5.** A1 put call on hold **6.** A1 resume the call **7.** A1 drop	Step 4- Check auto recording started on A1. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 5- Check A1 get CiscoTermConnRecordingEndEv Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check A1 get CiscoTermConnRecordingEndEv

Feature Interaction: Recording Use Cases

Scenario 13

FI: Redirect - IP phone (cluster3) call auto recording remote device (cluster1), redirect to IP phone (cluster3)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
• devRD (CTIRD3) has line RD1:9008 (active remote destination: 1711681 on cluster2) configured as:
  • GW preferred
  • automatic recording enabled
 	
Cluster 2:
• Phone D (RDD) has line D1 (1681)
 	
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• Phone B (SEPDB17) has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe A1, B1 in Prov3, D1 in Prov2, RD1 in Prov1
3. A1 (cluster3) call RD1 (cluster1) (3601 call 1729008)
4. Remote destination on cluster 2 answer the call
5. A1 redirect to B1
6. B1 disconnect the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check RD1 get CiscoTermConnRecordingEndEv
	Step 5- Check auto recording restarted on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 14

FI: Redirect -devRD (cluster1/SME), RDD (cluster2), A(cluster3/leaf) and B (cluster3/leaf) - RD auto recording and RD redirect

Setup and Action	Events and Call Info

Cluster 1:
• Central recorder: 2000
• devRD (CTIRD3) has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • automatic recording enabled
  • GW preferred
 	
Cluster 2:
• RDD has line RDD1 (1681)
 	
Cluster 3:
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devA (SEPDB17) has line A1 (2008) configured as:
  • Phone preferred
  • auto recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP3B5F) has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1 in Prov1, RDD1 in Prov2, A1, B1 in Prov3
3. A1 (cluster3) call RD1 (cluster1)
4. Remote destination on cluster 2 answer the call
5. RD1 redirect to B1 (cluster3)
6. B1 answer	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 4- Check auto recording started on A1. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = branch recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of A .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster3 Step 5: Check Recording retriggered on A1 Check A1 get CiscoTermConnRecordingEndedEv. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = branch recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of A

Setup and Action	Events and Call Info
	.getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster3

Scenario 15

FI: Redirect- devRD (leaf), RDD (SME), A (SME) and B (cluster2) - RD auto recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1729000 on cluster1) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 1.
• devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP723F) on cluster 2 has line B1 (4008) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1 in Prov1, RDD1 in Prov2, A1, B1 in Prov3
3. A1 call RD1
4. Remote destination answer the call
5. A1 redirect to B1
6. B1 answer
7. Close RD1 and reopen RD1 (unobserved and reobserve RD1)
8. RD1 disconnects the call	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 6- Recording retriggered on RD1. Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check RD1 CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() =

Setup and Action	Events and Call Info
	.getMediaForkingClusterID() = name of cluster1 Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 16

FI: Redirect- devRD (SME), RDD (leaf), A (leaf) and B (cluster2) - RD silent recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD4) on cluster 1 has line RD1 (9008) (active remote destination: 1802303 on cluster3) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 3.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP723F) on cluster 2 has line B1 (4008) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start silent recording on RD1
6. A1 redirect to B1 thru GW (i.e. A1 redirect to 171XXXX)
7. B1 answer
8. Start app recording on RD1
9. App stop recording on RD1
	Step 5- Start recording should fail on RD1 (error = CTIERR_RESOURCE_NOT_AVAILABLE??)
	Step 8- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 17

FI: Transfer- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 setup transfer to B1
6. B1 answer
7. A1 complete transfer
8. B1 disconnects the call
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Complete Transfer. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 18

FI: Direct Transfer- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A direct transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 call B1
6. B1 answer
7. App send direct transfer request on A1 with primary call = A1-RD1
8. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 19

FI: Conference- devRD (leaf), RDD (cluster2), A (SME) and B (SME) - RD silent recording and A conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start silent recording on RD1
6. A1 setup conference to B1
7. B1 answer
8. A1 complete conference
9. Close line RD1 and reopen line RD1
10. App stop recording on RD1
11. RD disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete conference. A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1.
	Step 9- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 10- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 20

FI: Join calls - devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A join calls

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 call A1
6. A1 answer
7. A1 join two calls wit primary call = A1 –RD1 call
8. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1.
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 21

FI: JAL- devRD (SME), RDD (cluster2), A1, A2 and B (leaf) - RD silent recording and A1 does JAL

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) and A2 (1623) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP3B5F) on cluster 3 has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start silent recording on RD1
6. A2 call B1
7. B1 answer
8. A1 join two calls with primary call = A1 RD1
9. Stop recording on RD1
10. RD1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- A2 and B1 are connected.
	Step 8- JAL. A1, B1 and RD1 in conference. Recording *not* retriggered on RD1.
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 22

FI: Drop any party- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A drop B from conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 set up conference to B1
6. B1 answer
7. A1 complete conference
8. Reopen RD1
9. A drop B from conference
10. RD1 disconnects the call	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- A1 complete conference. A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1. Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 9- A drop B. A1 and RD1 are connected as two parties call. Recording retriggered on RD1. Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

Setup and Action	Events and Call Info
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 10- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 23

FI: EM- devRD (leaf), RDD (cluster2), A (SME) - RD auto recording and A EM Login

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPAB21) on cluster 1 has line A1 (9000) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• EM profile with line E1 (7788)
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. EM login to devA with line E1
4. E1 call RD1
5. Remote destination answer the call
6. E1 disconnect call
7. EM logout
	Step 5- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- EM logout succeed.
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 24

Hunt list - devRD (leaf), RDD (cluster2), A (SME), B (SME) and HP(SME) - RD selective recording

Setup and Action	Events and Call Info
Cluster 1 (SME): • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: Cluster 3 (leaf): • "Allow trunk use GW recording" is enabled • Branch recorder: 2000 • devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as: • selective recording enabled • GW preferred Remote Destination RDD on cluster 2. • devA (IP9) on cluster 1 has line A1 (2302) configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 • devB (RP) on cluster 1 has line B1 (1500) configured as: • no recording • Cluster 1 (SME): • Line Group: LG2: A1 and B1, top down • Hunt list: HL2:LG2 • Hunt pilot: 3636 - HL2 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly **3.** RD1 call HP 3636 (1723636) **4.** After remote destination answer, call is offered to A1 **5.** A1 answer **6.** App start recording on RD1 **7.** App stop recording on RD1 **8.** RD1 disconnects the call	Step 5- Check recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 25

FI: Call Park- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording and A park and park reversion

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• Call park number P1 on same cluster of A with park reversion number set to A1
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start recording on RD1
6. A1 park call to park number P1
7. Wait for park reversion happen
8. A1 answer the call
9. App stop recording on RD1
10. A1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Recording *not* retriggered on RD1.
	Step 7- Park Reversion happens. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 answers. Recording remains.
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 26

FI: Barge- devRD (SME), RDD (cluster2), A (leaf), A' (leaf) - RD selective recording and A' does Barge

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2010) configured as:
  • GW preferred
  • selective recording enabled
• devA' (SEP1396) on cluster 3 has line A1' (2010) configured as:
  • GW preferred
  • selective recording enabled
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1’ does barge
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- A1' barge in succeeds. Check barge events.
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 27

FI: Dpark- devRD (leaf), RDD (cluster2), A (SME), B (SME) - RD selective recording and A dpark and B retrive

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• dpark DN: D1 (7001) is in same cluster of A, a prefix of 666 is to retrieve.
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start recording on RD1
6. A1 transfer call to dPark number D1
7. B1 calls dpark D1
8. Stop recording on RD1
9. B1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Recording *not* retriggered on RD1.
	Step 7- B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 28

FI: Monitoring- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD selective recording and B start monitoring

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP3B5F) on cluster 3 has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB is the Supervisor
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 start monitoring on A1
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- Check A1 get CiscoTermConnMonitoringStartEv.
	A1: CiscoTermConnMonitorTargetInfoEv
	B1: CiscoTermConnMonitorTargetInfoEv
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv
	Check A1 get CiscoTermConnMonitoringEndEv.

Scenario 29

FI: Whisper coaching- devRD (leaf), RDD (cluster2), A (SME) and B (SME) - RD selective recording and B start whisper coaching

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SIP2) on cluster 1 has line A1 (9001) configured as:
  • no recording
• devB (SIP3) on cluster 1 has line B1 (9002) configured as:
  • no recording
• devB is the Supervisor
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 start whisper coaching on A1
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- Check A1 get MonitoringStartEvent.
	A1:CallAttributeInfoEvent with type = coaching
	B1:CallAttributeInfoEvent with type = coaching
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 30

FI: Agent Greeting- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD selective recording and B start agent greeting

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (CTI1) on cluster 3 has line B1 (1600) configured as:
  • no recording
• devB is the IVR
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start agent greeting on A1 with B1 as IVR DN
6. App start recording on RD1
7. App stop recording on RD1
8. A1 disconnects the call
	Step 5- Check A1 get CiscoMediaStreamStartedEv (CTI: sendMediaToBIBStartEvent).
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 31

FI: Persistent connection- devRD (SME), RDD (cluster2), A (leaf) - RD auto recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. App makes CreatePersistentConnection() request on RD1
4. A1 call RD1
5. Remote device answer
6. RD1 disconnects the call
	Step 3- Persistent connection is created on RD1.
	Step 5- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 32

FI: Call pick up- devRD (SME), RDD (cluster2), A (leaf), B (leaf)- RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (CTI1) on cluster 3 has line A1 (1600) configured as:
  • no recording
• devB (SIP1) on cluster 3 has line B1 (9000) configured as:
  • no recording
• A1 and B1 are in pick up group: PG_1
• Service parameter: auto pick up enabled
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call B1
4. After remote destination answer, call is offered to B1 and A1
5. A1 pick up the call
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- After pickup, A1 and RD1 are connected.
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 33

FI: CTI Failover- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD (CTI Port) on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devRD in device pool DPPS1 (ccm1 (cluster1 PUB), ccm2 (cluster1 SUB))
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call A1
4. Remote destination answers, and then A1 answer
5. App start recording on RD1
6. Stop CTI Manager service on Pub1
7. Open provider on Sub1 (SME/Cluster1)
8. Reopen RD1
9. App start recording on RD1
10. RD1 disconnects the call
11. Start CTI Manager service on Pub1
	Step 4- A1 and RD1 are connected.
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Stop CTI Manager service on Pub1 successfully.
	Step 7- Open provider on Sub1 successfully.
	Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 34

FI: CPN- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD (CTI Port) on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2010) configured as:
  • Phone preferred
  • selective recording enabled
  • rec_profile1: 2000
• devRD in device pool DPPS1 (ccm1 (cluster1 PUB), ccm2 (cluster1 SUB))
• SIP trunk configured for CPN on SME/cluster1 and leaf, RD set up CPN also.
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call A1
4. Remote destination answers, and then A1 answer
5. App start recording on RD1
6. Stop CTI Manager service on Pub1
7. Open provider on Sub1
8. Reopen RD1
9. App start recording on RD1
10. RD1 disconnects the call
11. Start CTI Manager service on Pub1
	Step 4- A1 and RD1 are connected.
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Stop CTI Manager service on Pub1 successfully.
	Step 7- Open provider on Sub1 successfully.
	Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 35

FI: Redirect- local- devRD (leaf), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. A1 redirect to B1
6. B1 answer
7. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- A1 redirects.
	Step 6- B1 answers. B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 36

FI: Transfer- local-devRD (SME), RDD (cluster2), A (SME) and B (SME) - RD selective recording and A transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. App start user recording on RD1
6. A1 setup transfer to B1
7. B1 answer
8. A1 complete transfer
9. Stop recording on RD1
10. RD1 disconnects the call
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete transfer. B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 37

FI: Conference- local-devRD (SME), RDD (cluster2), A (SME) and B (SME) - RD selective recording and A conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. App start user recording on RD1
6. A1 setup transfer to B1
7. B1 answer
8. A1 complete transfer
9. App stop recording on RD1
10. RD1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete conference. A1, B1 and RD1 are in conference. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Recording Fail Event

Scenario 38

Failure Event: Hold Resume- auto recording on IP phone - devRD, A, C, D (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  - selective recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2206) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, C1, and D1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. A1 put call on hold
6. C1 call D1, D1 answer
7. A1 resume

Step 4- Check auto recording started on A1.

Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check A1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- A1 put call on hold. Check A1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 7- A1 resume. No CiscoTermConnRecordingFailedEv.

Step 8- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 10- Check auto recording started on A1.

Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check A1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

8. C1 drop **9.** A1 put on hold **10.** A1 resume **11.** A1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 11- Check A1 get CiscoTermConnRecordingEndedEv

Scenario 39

Failure Event: Redirect- auto recording on CTI remote device - devRD, A, C, D, B, E (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  - auto recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  - GW preferred
  - selective recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)
- devB (SEPAB21) on cluster 1 has line B1 (9000)
- devE (IP9) on cluster 1 has line E1 (2302)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, B1, C1, D1, and E1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers
5. Drop call from recorder
6. C1 call D1, D1 answer

Step 4- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- Drop call from recorder. Check RD1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 7-. A1 redirect and B1 answer. No CiscoTermConnRecordingFailedEv.

Step 9- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 10- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

7. A1 redirect to B1 **8.** B1 answer **9.** C1 drop **10.** B1 redirect to E1 **11.** E1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 11- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 40

Failure Event: Transfer- auto recording on CTI remote device - devRD, A, C, D, B, E (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  - auto recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  - GW preferred
  - selective recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)
- devB (SEPAB21) on cluster 1 has line B1 (9000)
- devE (IP9) on cluster 1 has line E1 (2302)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, B1, C1, D1, and E1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers
5. Drop call from recorder
6. C1 call D1, D1 answer

Step 4- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- Drop call from recorder. Check RD1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 9-. A1 complete transfer. No CiscoTermConnRecordingFailedEv.

Step 10- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 13- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

7. A1 setup transfer to B1 **8.** B1 answer **9.** A1 complete transfer **10.** C1 drop **11.** B1 transfer to E1 **12.** E1 answer **13.** B1 complete transfer **14.** E1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 14- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 41

Cluster ID in Open Provider: Open providers after changing cluster ID

Setup and Action	Events and Call Info
Cluster 1 (SME): • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: Cluster 3 (leaf): • "Allow trunk use GW recording" is enabled • Branch recorder: 2000 Remote Destination RDD on cluster 2. **1.** Open provider on cluster 3 Leaf **2.** Change cluster ID on cluster 3 (e.g. “ClusterArcadia2013”) **3.** Restart CTI and CCM services **4.** Provider is reopened on cluster 3 Leaf **5.** Change cluster ID on cluster 3 back to original (e.g. “ClusterID3”) **6.** Restart CTI and CCM services **7.** Provider is reopened on cluster 3 Leaf **8.** Close provider	Step 4- After provider is reopened, check to see new cluster ID via VerifyProviderInfo on getClusterID. Step 4- After provider is reopened, check to see original cluster ID via VerifyProviderInfo on getClusterID.

Secured Recording

Secured Recording Use Cases

Scenario	Expected Result	Info
Scenario 1 1 .Agent and recorder are encrypted 2. Customer is in a secured/non-secured call with the agent. 3. The agent initiates a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv TermA CiscoTermConnRecordingTargetInfoEv	
Scenario 2 1. Agent is non-secured and recorder is encrypted 2. Customer is in a non-secured call with agent 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv TermA CiscoTermConnRecordingTargetInfoEv	
Scenario3 1. Agent is encrypted and recorder is non-secured 2. Customer is in a secured/non-secured call with the agent 3. The agent issues a request to record the call	CallCtlTermConnTalkingEv TermA CiscoTermConnRecordingStartEv CiscoTermConnRecordingEndEv	Cause = CAUSE_BCNAUTHORISED
Scenario 4 1. Agent and Recorder are non-secured 2. Customer and Agent are in a non-secured call 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv termA CiscoTermConnRecordingTargetInfoEv	
Scenario 5 1. Agent and recorder are authenticated 2. Agent and customer are in a call 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request fails with exception	

Monitoring and Recording Use Cases

Scenario	Expected Result
Scenario 1 1. Agent is non-secured, Supervisor and recorder are encrypted 2. Agent is in a non-secured call with customer 3. Supervisor monitors the call in non-secured mode	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS
4. Application requests for recording the call at Supervisor	CiscoTermConnRecordingStartEv CallCtlTermConnTalkingEv TermS The recording is done in a secured mode as Both the supervisor and recorder are secured, but the overall call security is non-secured.

Redirect Set OriginalCalledID

The following scenario illustrates the message flows for Redirect Set OriginalCalledID.

Scenario One

- A, B, and C appear in an applications controlled list.
- D is does not appear in the control list.
- A calls B.
- B redirects call to D with C as preferredOriginalCalledParty.

Application will see following events for parties A and B:

Meta Event Cause	Call	Event	Fields
META_CALL_ADDING_PARTY	Call 1	ConnCreatedEv for D ConnConnectedEv for DCallCtl ConnEstablishedEv for D	CallingParty = A CalledParty = B **LastRedirectedParty = C** CurrentCalledParty = D
META_CALL_REMOVE_PARTY	Call 1	ConnDisconnectedEv for B CallCtlConnDisconnectedEv for B TermConnDroppedEv for B CallCtlTermConnDroppedEv for B CallObservationEndedEv for B	CallingParty = A CalledParty = B **LastRedirectedParty = C** CurrentCalledParty = D

Note The specified event group may not be in the same order and might change depending on where parties are present in the cluster, on the load, and other conditions.

Scenario Two

- A, B, and C do not appear in the Control list, and
- D is in the application control list.
- A calls B.
- B redirects the call to D with C as preferredOriginalCalledParty.

The application will see following events for party D:

.

Meta Event Cause	Call	Event	Fields
META_CALL_STARTING	Call1	CallActiveEv ConnCreatedEv for D ConnInProgressEv for D CallCtlConnOfferedEv for D ConnCreatedEv for A CallCtlConnInitiatedEv for A	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D
META_CALL_PROGRESS	Call1	ConnAlertingEv for D CallCtlConnAlertingEv for D TermConnCreatedEv for D CallCtlTermConnRingingEv for D ConnConnectedEv for A CallCtlConnEstablishedEv for A	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D
META_CALL_PROGRESS	Call	ConnConnectedEv for D CallCtlConnEstablishedEv for D TermConnActiveEv for D CallCtlTermConnTalkingEv for D	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D

Redirect to a Device

The following tables display message sequence charts for the Redirect enhancement that allows you to redirect calls to a specific device, even if that device is sharing a line with another device.

CallRedirect to Shared Line with Device Name

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls B and B answers	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B	

Action	Events	CallInfo
Application redirects the call from B to C using CiscoConnection.redirect() method with devcieName as C	GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 CallCtlConnOfferedEv C GC1 ConnAlertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEvImpl TermC GC1 TermConnCreatedEv TermC' GC1 TermConnPassiveEv TermC' GC1 CallCtlTermConnInUseEv TermC' GC1 TerConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT

CallRedirect to Shared Line with Invalid Device Name

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls B and B answers	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B	
Application redirects the call from B to C using CiscoConnection.redirect() method with deviceName as D		InvalidPartyException caught: geterrorcode() = CiscoJtapiException. REDIRECT_CALL_INVALID _DEVICE_NAME

CallRedirect to Shared Line using selectRoute

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls route point RP. RP redirects the call to the destination C using selectRoute() method	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv RP GC1 ConnInProgressEv RP GC1 CallCtlConnOfferedEv RP	
RP redirects the call to the destination C using selectRoute() method	GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 ConnAltertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEv TermC GC1 ConnCreatedEv C' GC1 TermConnPassiveEv TermC' GC1 CallCtlTermConnInUseEv TermC'	CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT

Verify Remote Destination Support

Table 346: Verify Remote Destination in Add Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("testRDD", "77000", true)	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestinationInfo[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationName() = "testRDD" CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "77000" CiscoRemoteDestination Info[0].getIsActiveRD() = true

Table 347: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("77000", "testRDD", "79000", true)	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestinationInfo[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationName() = "testRDD" CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "79000" CiscoRemoteDestination Info[0].getIsActiveRD() = true

Table 348: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Number("77000", "79000")	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestination Info[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "79000"

Table 349: Verify Remote Destination in Add Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("testRDD", "99000", true)	Caught exception com.cisco.jtapi.PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_ CONNECT_DESTINATION _NOT_REACHABLE.

Table 350: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("77000", "testRDD", "99000", true)	Caught exception com.cisco.jtapi. PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_CONNECT _DESTINATION_NOT_REACHABLE.

Table 351: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestinationNumber()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestinationNumber ("77000", "99000")	Caught exception com.cisco.jtapi. PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_CONNECT _DESTINATION_NOT_REACHABLE.

Secure Conferencing

Action	Events	Call info
Scenario:1 Configuration: A (secure) and B (secure). A calls B. B answers. Application issues getCallSecurtyStatus().	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 Returns the call security status of the call.	Calling: A Called: B CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info. CallSecurityStatus = 3 (ENCRYPTED).
Configuration: A (secure), B (secure) and C (non-secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv () A calls B. B answers. B call C. C answers. B completes conference.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1.	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED). Note: Though CallSecurityStatus = NotAuthenticated, A and B will continue to have secure media between themselves and the conference bridge, i.e. they will continue to receive SRTP key info because they are encrypted parties.

Action	Events	Call info
Scenario:3 Configuration: A (secure), B (secure) and C (secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv () A calls B. B answers. B call C. C answers. B completes conference. Application issues getCallSecurtyStatus().	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1. Returns the call security status of the call (Secure).	Participants: A, B, C CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info.
Scenario:4 Application does not add call observers on A, B, C. Configuration: A (secure), B (secure) and C (non-secure). A calls B. B answers. B call C. C answers. B completes conference. Application later adds call observers on A, B, C. Application issues getCallSecurtyStatus().	CallSecurityStatusChangedEv for callID = GC1 with Cause = CAUSE_SNAPSHOT Returns the call security status of the call.	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info.
Scenario:5 Configuration: A (secure), B (secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A calls B. B answers. B puts call on hold. B resumes call.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 CallSecurityStatusChangedEv for callID = GC1 CallSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info. CallSecurityStatus = 0 (UNKNOWN) will get updated in the call info. CallSecurityStatus = (ENCRYPTED) will get updated in the call info.

Action	Events	Call info
Scenario:6 Configuration: A (secure), B (secure) and C (non-secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A calls B. B answers. B consults C. C answers. B completes transfer.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC2 CallCtlSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info for GC1. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC2. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC1.

Action	Events	Call info
Scenario:7 Configuration: A (secure), B (secure), C (secure), D (secure), and E (Authenticated). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A, B and C are part of a conference Call 1. C, D and E are a part of another conference Call 2. C chains the 2 conferences.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for C' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for D' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for E' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallCtlSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info for GC1. CallSecurityStatus = 2 (AUTHENTICATED) will get updated in the call info for GC2. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC1 and GC2.
Scenario:8 Configuration: A (secure), B (secure), B (authenticated) Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv()	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1.	CallSecurityStatus = 2 (AUTHENTICATED) will get updated in the call info for GC1. **Note** Applications who have added call observers on B' will also get the event, i.e. the new event will be delivered to RIU Parties as well.

Secure Connection Enhancements

Action	Events	Call information
The application wants to connect securely to Cisco Unified Communications Manager and downloads the certficate using the interfaces in CiscoJtapiProperties. After down loading the certificate, application initializes provider using a providerString formatted with new parameters: String providerString = serverName + ";login = " + login + ";passwd = " + passwd + ";InstanceID = " + instanceID + ";CAPF = " + CAPFserver + ";CAPFPort = " + capfport + ";TFTP = " + TFTPServer + ";TFTPPort = " + tftpport + ";CertPath = " + certificatepath+ ";CertStorePassphrase = " + cartificatepassphrase +";" JtapiPeer peer = JtapiPeerFactory.getJtapiPeer (null); MyProviderObserver providerObserver = new MyProviderObserver (); provider = peer.getProvider (providerString);	Provider is initialized correctly through TLS connection. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals ProvInService event is delivered to provider observer	
The application is not interested in secure connection and open provider using userid and passwd in provider String String providerString = serverName + ";login = " + login + ";passwd = " + ";"	Provider is initialized correctly. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals ProvInService is delivered to provider observer	
The application uses jtprefs to download the certificates and initializes provider specifying only the userid, passwd, instanceid and certificate pass phrase in provider string. String providerString = serverName + ";login = " + login + ";passwd = " + passwd + ";InstanceID = " + instanceID +":CertStorePassphrase = " + cartificatepassphrase +";"	Provider is initialized correctly through TLS connection. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals	

Secure Icon Enhancements

Enable the callSecurityStatusChangedEv using JTAPI ini parameters or using the JTAPIProperties.

Cluster1 and Cluster2 are secured and User is also a secured user having a CAPF profile associated with it. Enable "SRTP Allowed" in the SIP trunk Configurations.

TermA is registered to Cluster1 with address A.

TermB is registered to Cluster2 with address B

TermC is registered to Cluster2 or Cluster1 as per Use case and has address C

SIP trunk is configured on Cluster1 to make calls to cluster2

A Route Pattern is configured on Cluster 1 to route the call to the SIP trunk

Use Case One

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1. Security Mode of Both term A and termB is encrypted. Security mode of the SIP trunk is non secured.		
A calls B though the route pattern.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	
Call is offered on B and B accepts.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Two

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1. Security Mode for term A and termB and term C is encrypted. Security mode of the SIP trunk is non secured.		
A calls B though the route pattern.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	

Action	Expected results	Information
Call is offered to B and B accepts.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B answers the call.	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED
B Redirects the call to C	TermA CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermA CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 CallCtlConnOfferedEv C GC1 ConnAlertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEvImpl termC GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv TernB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
C answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Three

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1 Security Mode of Both term A and termB and term C is encrypted Security mode of the SIP trunk is non secured		
A calls B though the route pattern	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	

Action	Expected results	Information
Call is offered on B and B accepts	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED
B calls C	GC1 CallCtlTermConnHeldEv TermB GC1 CiscoCallSecurityStatusChangedEv GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConenctedEvB GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TermB GC2 TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 CallCtlConnDialingEv B GC2 ConnEstablishedEv B	Ev.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
Call is offered on C and C accepts	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv TermC GC2 TermConnRingingEv TermC GC2 CallCtlTermConnRingingEvImpl termC	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
C answers the call	TermB CiscoRTPOutputStartedEv GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv TermC TermB CiscoRTPInputStartedEv TermC CiscoRTPOutputStartedEv TermC CiscoRTPInputStartedEv GC2 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Action	Expected results	Information
B does a Direct Transfer GC1.transfer(GC2)	GC1 CiscoTermConnSelectChangedEv termB GC2 CiscoTermConnSelectChangedEv TermB TermC CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermC CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnEstablishedEv C GC1 TermConnCreatedEv TermC Gc1 TermConnActiveEv TermC Gc1 CallCtlTermConnTalkingEv TermC GC2 TermConnDroppedEv TermC GC2 CallCtlTermConnDroppedEv TermC GC2 ConnDisconnectedEv C GC2 CallCtlConnDisconnectedEv C GC2 TermConnDroppedEv termB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectedEv B GC2 CallInvalidEv GC1 TermConnDroppedEv termB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CiscoTransferEndEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Four

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When using both sRTP and TLS' in the SIP trunk config on cluster1 Security Mode of Both term A and termB and term C is encrypted Security mode of the SIP trunk is non secured		
A calls B though the route pattern	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	
Call is offered on B and B accepts	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B calls C	GC1 CallCtlTermConnHeldEv TermB GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConenctedEvB GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TermB GC2 TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 CallCtlConnDialingEv B GC2 ConnEstablishedEv B	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
Calls is offered on C and C accepts	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv TermC GC2 TermConnRingingEv TermC GC2 CallCtlTermConnRingingEvImpl termC	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

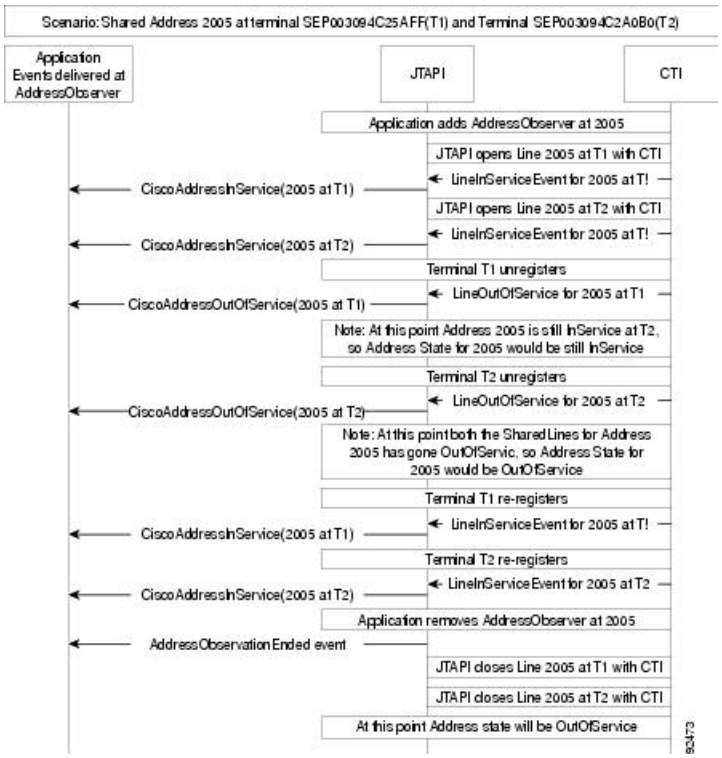
Action	Expected results	Information
C answers the call	TermB CiscoRTPOutputStartedEv GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv TermC TermB CiscoRTPInputStartedEv TermC CiscoRTPOutputStartedEv TermC CiscoRTPInputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B does a Direct Transfer GC1.transfer(GC2)	GC1 CiscoTermConnSelectChangedEv termB GC2 CiscoTermConnSelectChangedEv TermB TermC CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermC CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnEstablishedEv C GC1 TermConnCreatedEv TermC Gc1 TermConnActiveEv TermC Gc1 CallCtlTermConnTalkingEv TermC GC2 TermConnDroppedEv TermC GC2 CallCtlTermConnDroppedEv TermC GC2 ConnDisconnectedEv C GC2 CallCtlConnDisconnectedEv C GC2 TermConnDroppedEv termB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectedEv B GC2 CallInvalidEv GC1 TermConnDroppedEv termB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CiscoTransferEndEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

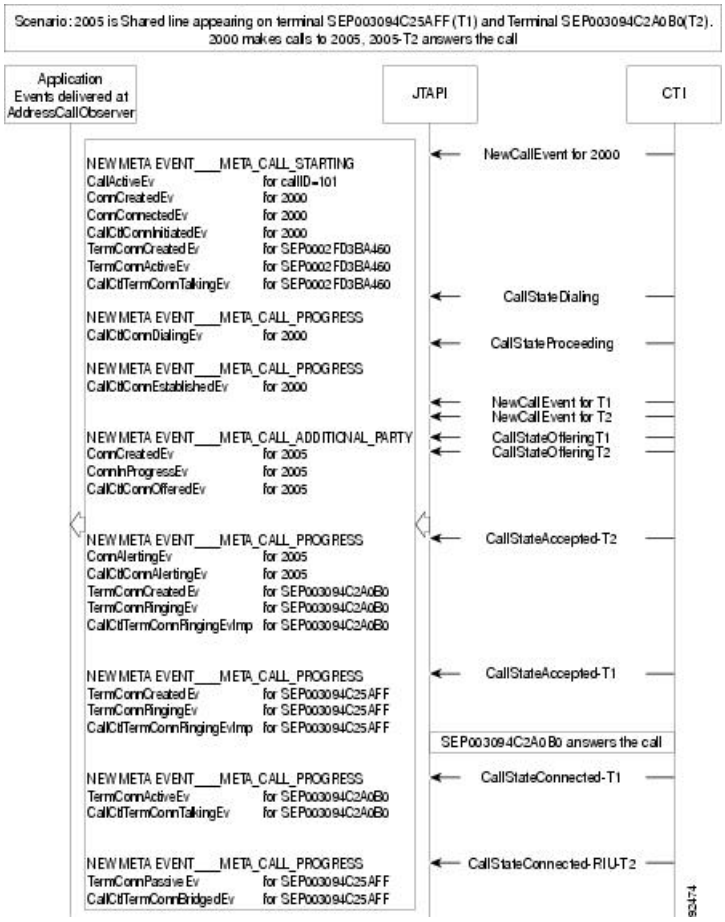
Shared Line Support

The following diagrams illustrate the message flows for Shared Line support.

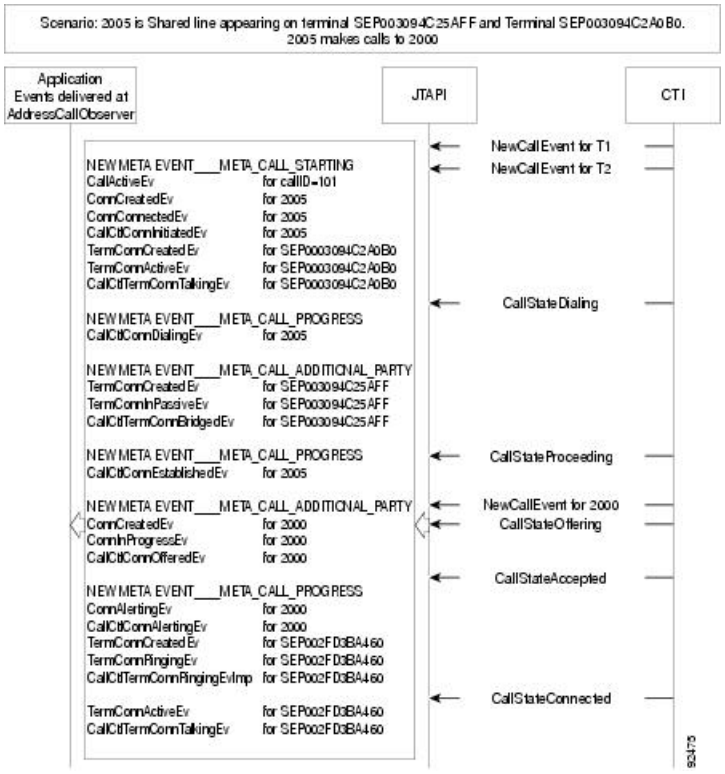
AddressInService/AddressOutofService Events



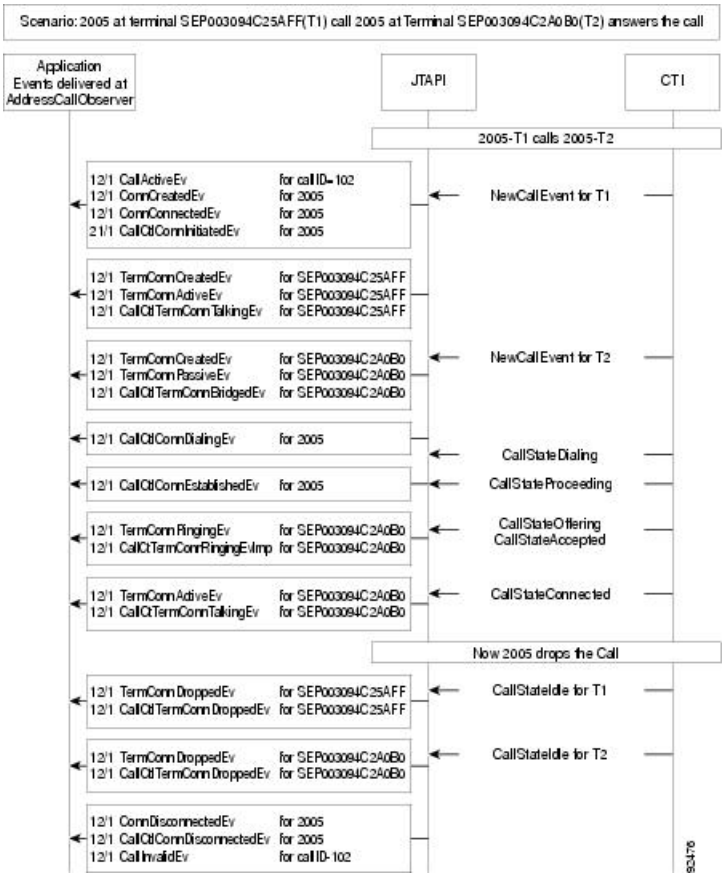
Incoming Call to Shared Address



Outgoing Call From Shared Address



Shared Address Calling Itself



Single Sign-On

Here are the list of use cases for this feature.

Sl. No	Scenario	Result
1.	Start the application (cucimoc) and getProvider(str) API is called by application specifying the singlesignon ticket. Application calls getAddresses() API.	Returns provider to the application and all the addresses configured in the control list.
2.	Application specifies an invalid ticket but correct userid and password in API.	Throws a platform exception to getProvider() API.

Sl. No	Scenario	Result
3.	Start the application and it calls the getProvider() API with singlesignon ticket. The network connectivity is lost. Application gets a new singlesignon token and calls getProvider().	Returns provider to the application. Delivers ProvOutOfServiceEv to provider observer. JTAPI connects and tries to authenticate the user which fails. Delivers ProvShutdownEv to provider observer. Returns provider object to the application.
4.	Start the application and getProvider() API is called by application with the singlesignon ticket. But the feature is not enabled on Cisco Unified Communications Manager.	Throws PlatformException and getErrorCode() returns CiscoJTAPIException - CTIERR_SSO_DISABLED.
5.	Start the application and getProvider() API is called by application with invalid singlesignon ticket.	Throws PlatformException and getErrorCode() returns CiscoJTAPIException - CTIERR_DIRECTORY_LOGIN_FAILED.
6.	Multiple providers: Start the application and getProvider() on two nodes in the cluster with the same token.	Both getProvider() call is successful with the first provider. Throws exception to the second getProvider().

Single Step Transfer

Addresses A, B, and C appear in the control list, and the call between A and B is then gets transferred to C with B as the transfer controller. Applications will see the following events:

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string)	ConnCreatedEv 5003 Cause: CAUSE_NORMAL ConnInProgressEv 5003 Cause: CAUSE_NORMAL CallCtlConnOfferedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnAlertingEv 5003 Cause: CAUSE_NORMAL CallCtlConnAlertingEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	NEW META EVENT__META_CALL_ REMOVING_PARTY TermConnDroppedEv CTIP2 Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv CTIP2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnDisconnectedEv 5002 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5002 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv 5003 Cause: CAUSE_NORMAL ConnInProgressEv 5003 Cause: CAUSE_NORMAL CallCtlConnOfferedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnCreatedEv 5001Cause: CAUSE_NORMAL ConnConnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnEstablishedEv 5001Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string) (continued)	CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 5003 CAUSE_NORMAL CallCtlConnEstablishedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL	NEW META EVENT_____META_UNKNOWN CallObservationEndedEv Cause: CAUSE_NORMAL	ConnAlertingEv 5003 Cause: CAUSE_NORMALCallCtl ConnAlertingEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv CTIP3 TermConnRingingEv CTIP3Cause: CAUSE_NORMAL CallCtlTermConnRingingEvImpl CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 2004 Cause: CAUSE_NORMAL CallCtlConnEstablishedEv 5003Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string) (continued)			TermConnActiveEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStoppedEv Cause: CAUSE_NORMAL CiscoRTPOutputStoppedEv Cause: CAUSE_NORMAL ConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL ConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL META_UNKNOWN CallInvalidEv [#32] Cause: CAUSE_NORMAL

SIP REPLACE

For the JTAPI events in the scenario described below, we have not shown Terminal events. It will be sent for all the observed Terminals as usual. Also events are shown with the assumption that only A, B, or C is observed; events would vary if combination of A, B, or C is observed.

SN	Scenario	Events at A	Events at B	Events at C
1.	REPLACE with INVITE a confirmed Dialog: A (Dialog1) is in Call with B (Dialog2) (GC1). C sends INVITE with REPLACE Dialog2 (GC2). After replace is completed, A (Dialog1) and C (Dialog3) are in the Call	GCID and CPIC with reason REPLACES, Cgpn = C, Cdpn = A, Ocdpn = A, Lrp = B JTAPI Events: CiscoCallChangedEv - (GC1 -GC2) ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B–GC1 ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A–GC1 CallInvalid -GC1 CallActive -CG2 ConnCreatedEv –C -GC2 ConnConnectedEv –C -GC2 CallCtlConnEstablishedEv -C -CG2 ConnCreatedEv –A –GC2 ConnConnectedEv –A -GC2 CallCtlConnEstablishedEv -A -CG2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B	CSCE IDLE with reason REPLACES JTAPI Events: ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv -A -GC1 ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B -GC1 CallInvalidEv -GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES	NewCall/CSCE -Dialing/ CSCE -Connected with Cgpn = C, Cdpn = A, Ocdpn = B, Lrp = B JTAPI Events: CallActiveEv –GC2 ConnCreatedEv –C –GC2 ConnConnectedEv -C -–GC2 CallCtlConnEstablishedEv -C–GC2 ConnCreatedEv –A–GC2 ConnConnectedEv A -–GC2 CallCtlConnEstablishedEv -A -–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B

SN	Scenario	Events at A	Events at B	Events at C
2.	REPLACE with INVITE an early Dialog: A (Dailog1) is in Call with B (Dialog2) (GC1), B is ringing. C sends INVITE with REPLACE Dialog2 (GC2). After replace completed, A (Dialog1) and C (Dialog3) in the Call	GCID and CPIC with reason REPLACES, Cgpn = C, Cdpn = A, Ocdpn = A, Lrp = B JTAPI Events CiscoCallChangedEv - (GC1 -GC2) ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B–GC1 ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A–GC1 CallInvalid -GC1 CallActive -CG2 ConnCreatedEv –C -GC2 ConnConnectedEv –C -GC2 CallCtlConnEstablishedEv -C -CG2 ConnCreatedEv –A –GC2 ConnConnectedEv –A -GC2 CallCtlConnEstablishedEv -A -CG2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B	CSCE -Idle with reason REPLACES JTAPI Events: ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv -A -GC1 ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B -GC1 CallInvalidEv -GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES	NewCall/CSCE -Dialing/CSCE -Connected, with Cgpn = C, Ccdpn = A, Ocdpn = A, Lrp = B JTAPI Events: CallActiveEv –GC2 ConnCreatedEv –C –GC2 ConnConnectedEv -C -–GC2 CallCtlConnEstablishedEv -C–GC2 ConnCreatedEv –A–GC2 ConnConnectedEv A -–GC2 CallCtlConnEstablishedEv -A -–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B

SN	Scenario	Events at A	Events at B	Events at C
3.	REPLACE with INVITE an early Dialog: A (Dialog1) is in Call with B (Dialog2) (GC1), B is ringing. C sends invite with replace Dialog -X (GC2)			NewCall/CSCE_Dialing/ reason REPLACES CSCE -Disconnected with reason REPLACES JTAPI Events: CallActiveEv –GC2 ConnCreatedEv –C–GC2 ConnConnectedEv –C–GC2 CallCtlConnEstablishedEv -C -–GC2 ConnFailedEv –C -–GC2 ConnConnectedEv –C -–GC2 CallCtlConnEstablishedEv -A -–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = , CurrentCalling = C, CurrentCalled = , LastRedirecting =
4.	REFER request with REPLACE Dialog: When REPLACE Dialog is in a Cisco Unified Communications Manager Cluster. A is in call with B (REFEREE) Dialog1, and Dialog2 A is in Call with C (REFER TO TARGET) Dialog3 and Dialog4 SIP -UA A send REFER B on Dialog1 to C with REPLACES Dialog3	TransferStartEv CSCE -Idle at Dialog1 with reason TRANSFER and at Dialog3 with reason TRANSFER TransferEndEv JTAPI Event: Regular TransferEvent	TransterStartEv CPIC with reason TRANSFER and Cgpn = B, Cdpn = C, Lrp = A OCdpn = C TransferEndEv JTAPI Event: Regular TransferEvents	TransferStartEv GCID with reason TRANSFER and Cgpn = B, Cdpn = C, Lrp = A OCdpn = C TransferEndEv JTAPI Event: Regular TransferEvents

SN	Scenario	Events at A	Events at B	Events at C
5.	REFER request with REPLACE Dialog: When REPLACE Dialog is outside Cisco Unified Communications Manager Cluster SIP -UA A is in call with B, Dialog1 and Dialog2 (GC1) SIP -UA A is in call with SIP -UA C Dialog3 SIP -UA A sends REFER B on Dialog1 to SIP -UA C with REPLACES Dialog3	No Events	CPIC with reason REFER and Cgpn = B, Cdpn = C, Lrp = A OCdpn = B JTAPI Events: ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A -GC1 ConnCreatedEv – C –GC1 ConnConnectedEv –C -GC1 CallCtlConnEstablishedEv –C -GC1 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REFER JTAPI CallInfo: Calling = A, Called = B, CurrentCalling = B, CurrentCalled = C, LastRedirecting = A	No Events

SN	Scenario	Events at A	Events at B	Events at C
6.	REFER request with REPLACE Dialog: When A is outside a Cisco Unified Communications Manager Cluster SIP -UA A is in call with B, Dialog1 and Dialog2 SIP -UA A is in call with C Dialog3 and Dialog4 SIP -UA A sends REFER B on Dialog1 to C with REPLACES Dialog3	No Events	CPIC with reason REPLACES and Cgpn = B, Cdpn = C, Lrp = A, OCdpn = C JTAPI Events: ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A -GC1 ConnCreatedEv – C - GC1 ConnConnectedEv –C –GC1 CallCtlConnEstablishedEv –C -GC1 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = A, Called = B, CurrentCalling = B, CurrentCalled = C, LastRedirecting = A	GCID with reason REPLACES and Cgpn = B, Cdpn = C, Lrp = A OCdpn = C JTAPI Events: CiscoCallChangedEv (GC2 -GC1) ConnDisconnectedEv –A -GC2 CallCtlConnDisconnectedEv -A -GC2 ConnDisconnectedEv –C -GC2 CallCtlConnDisconnectedEv -C -GC2 CallInvalid -GC2 CallActive -CG1 ConnCreatedEv –B –GC1 ConnConnectedEv –B -GC1 CallCtlConnEstablishedEv -B -CG1 ConnCreatedEv –C -GC1 ConnConnectedEv –C -GC1 CallCtlConnEstablishedEv -C -CG1 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = A, Called = B, CurrentCalling = B, CurrentCalled = C, LastRedirecting = A

SN	Scenario	Events at A	Events at B	Events at C
7.	REFER request with REPLACE Dialog: When REPLACE Dialog is in a Cisco Unified Communications Manager Cluster. A is in call with B (REFEREE) Dailog1, and Dialog2 (GC1) D is in Call with C (REFER TO TARGET) Dialog3 and Dialog4 (GC2) A sends REFER B on Dialog1 to C with REPLACES Dialog3 B and C in final call.	CSCE -Idle at Dialog1 with reason REFER and at Dialog3 with reason REPLACES JTAPI Events: ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv -A -GC1 ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B -GC1 CallInvalidEv -GC1 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REFER Event at D: ConnDisconnectedEv –D –GC2 CallCtlConnDisconnectedEv -D -GC2 ConnDisconnectedEv –C -GC2 CallCtlConnDisconnectedEv -C -GC2 CallInvalidEv -GC2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES	CPIC with reason REPLACES and Cgpn = B, Cdpn = C, Lrp = D OCdpn = C JTAPI Events: ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A -GC1 ConnCreatedEv –C -GC1 ConnConnectedEv –C –GC1 CallCtlConnEstablishedEv -C -GC1 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = A, Called = B, CurrentCalling = B, CurrentCalled = C, LastRedirecting = D	GCID with reason REPLACES and Cgpn = B, Cdpn = C, Lrp = D, OCdpn = C JTAPI Events: CiscoCallChangedEv (GC2 -GC1) ConnDisconnectedEv -D CallCtlConnDisconnectedEv -D ConnDisconnectedEv -C CallCtlConnDisconnectedEv -C CallInvalid -GC2 CallActive -CG1 ConnCreatedEv –C -GC1 ConnConnectedEv –C -GC1 CallCtlConnEstablishedEv -C -CG1 ConnCreatedEv –B –GC1 ConnConnectedEv –B -GC1 CallCtlConnEstablishedEv -B -CG2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = C, CurrentCalling = B, CurrentCalled = C, LastRedirecting = D

SIP REFER

The following section describes the scenarios that might be encountered during a SIP REFER. There are two categories of REFER scenarios: IN-Dialog and OutOfDialog.

IN-Dialog REFER Scenario

There are 11 scenarios (A through K) described in the sections that follow for IN-Dialog REFERs.

Scenario One

A (SIP UA in cluster/in control) is in a call with B.

A (referrer) REFERs B (Referee) to C (Refer to target), C is Ringing.

JTAPI moves A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection into the "UNKNOWN" state.

CAUSE_CODE provided will be CAUSE_NORMAL, new API provides REASON_REFER.

For C a new Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection would be created.

CallInfo at B and C would be as follows:

At B: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C
At C: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

JTAPI Application observing B will see:

getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty() = B
getCurrentCalledParty() = C
getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B
getCalledParty() = C
getCurrentCallingParty() = B
getCurrentCalledParty() = C
getLastRedirecting() = A

Scenario Two

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C Answers the Call.

JTAPI will Disconnect/Drop A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API would provide REASON_REFER.

For C Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will move to the Connected/Established/Active/Talking state.

CallInfo at B and C will be as follows

At B: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C
At C: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Three

A(SIP UA inside cluster) is in a call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C is ringing but C did not answer the call and has no forward configured. Refer fails, the original call between A and B is restored.

JTAPI will Disconnect/Drop the Connection/CallControlConnection/TerminalConnection/

CallControlTerminalConnection for C. CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER and move A's Connection/CallControlConnection/

TerminalConnection/CallControlTerminalConnection from the "Unknown" state to the Connected/

Established/Active/Talking state.

CallInfo at A and B will be as follows

At A: Cgpn = A, Cdpn = B, Lrp = OCdpn = B

At B: Cgpn = A, Cdpn = B, Lrp = OCdpn = B

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

Scenario Four

A(SIP UA outside cluster) is in call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C is ringing.

JTAPI will create Connection/CallControlConnection/TerminalConnection/

CallControlTerminalConnection for C and will drop A's Connection/CallControlConnection on getting CPIC at B, CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER.

CallInfo at B and C will be as follows:

At B: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

At C: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Five

A(SIP UA outside cluster) is in a call with B.

A(referrer) refers B(Referee) to C(Refer to target), C is ringing but C did not answer the call and has no forward configured. Refer fails, the original Call between A and B is restored.

JTAPI will create Connection/CallControlConnection for A again and drops Connection/

CallControlConnection/TerminalConnection/CallControlTerminalConnection for C.

CAUSE_CODE provided will be CAUSE_NORMAL and new API will provide REASON_REFER.

CallInfo at A and B will be as follows

At A: Cgpn = A, Cdpn = B, Lrp = OCdpn = B

At B: Cgpn = A, Cdpn = B, Lrp = OCdpn = B

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

JTAPI Application observing C will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

Scenario Six

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C answers the call.

JTAPI moves Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for C to the Connected/Established/Active/Talking state. CAUSE_CODE provided is CAUSE_NORMAL and the new API will provide REASON_REFER.

CallInfo at B and C will be as follows

At B: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

At C: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Seven

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C forwardAll to D, D is ringing.

JTAPI creates Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for D. CAUSE_CODE provided will be CAUSE_REDIRECT and the reason received from CTI would be ForwardAll.

CallInfo at B and D will be as follows

At B: Cgpn = B, Cdpn = D, Lrp = C OCdpn = C

At D: Cgpn = B, Cdpn = D, Lrp = C OCdpn = C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

Scenario Eight

A (SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERs B(Referee) to C(Refer to target), C Redirect to D, D is ringing.

JTAPI creates Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for D. CAUSE_CODE provided will be CAUSE_REDIRECT and the reason received from CTI in NewCallEvent at D will be Redirect.

Callinfo when Call is offered at C:

At B: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

At C: Cgpn = B, Cdpn = C, Lrp = A OCdpn = C

CallInfo in final Call:

At B: Cgpn = B, Cdpn = D, Lrp = C OCdpn = C

At D: Cgpn = B, Cdpn = D, Lrp = C OCdpn = C

JTAPI Application observing B will see in final Call:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

Scenario Nine

A(SIP UA in cluster/in control) is in a call with B.

B consult transfer to D, A(Referrer) REFERs B(Referee) to C(Refer to target), C is ringing,

Bcompletes the transfer. Attempt to transfer will fail while C is ringing.

Scenario Ten

A(SIP UA in cluster/in control) is in a call with B.

B consult transfer to D, A(Referrer) REFERs B(Referee) to C(Refer to target), C answers the call.

Refer will be successful. B completes the transfer, transfer will be successful, C and D will be in call.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER.

For C, Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will move to Connected/Established/Active/Talking state.

CallInfo at D and C would be as follows

At D: Cgpn = C, Cdpn = D, Lrp = B OCdpn = D

At C: Cgpn = C, Cdpn = D, Lrp = B OCdpn = D

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = C

getCurrentCalledParty() = D

getLastRedirecting() = B

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = C

getCurrentCalledParty() = D

getLastRedirecting() = B

Scenario Eleven

B is in a call with D, B consults to A(SIP UA in cluster/in control).

A(Referrer) REFERs B(Referee) to C(Refer to target), C is ringing, B completes the transfer.

REFER would fail. Call at A will be dropped, transfer is successful, D is getting RingBack, C is ringing.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API would provide REASON_REFER, Application will not know if REFER failed.

For C, Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will move to Alerting/Alerting/Ringing/Ringing state.

CallInfo at D and C would be as follows:

At D: Cgpn = D, Cdpn = C, Lrp = B OCdpn = C

At C: Cgpn = D, Cdpn = C, Lrp = B OCdpn = C

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = D

getCurrentCalledParty() = C

getLastRedirecting() = B

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = D

getCurrentCalledParty() = C

getLastRedirecting() = B

OutOfDialog Refer

SIP-UA A REFERs B(Referee) to C (Refer To Target)

B gets newcall with Cgpn = A, Cdpn = B, Lrp = , OCdpn = B.

JTAPI Application will get CallActive, Connection, CallCtlConnection, TerminalConnecton and CallCtlTerminalConnection created for B with CAUSE_NORMAL, and the new API will return REASON_REFER.

B's Connection/CallCtlConnection, TerminalConnection/CallCtlTerminalConnection will go into the Connected/Established/Active/Talking state. JTAPI creates Connection and CallCtlConnection for A in "UNKNOWN" state based on FarEndPointType_ServerCall provided by CTI/CP.

B answers the call and is connected to A (at this point no RTPEvent will be sent).

B get CallPartyInfoChangedEv with Cgpn = B, Cdpn = C, Lrp = A, OCdpn = C, Reason = REFER.

C get NewCall offering with Cgpn = B, Cdpn = C, Lrp = A, OCdpn = C, Reason = REFER.

JTAPI Application will get Connection, CallControlConnection, TerminalConnecton and CallCtlTerminalConnection created for B with CAUSE_NORMAL, and the new API will return REASON_REFER.

C Accepts/Answers the call, B is connected to C (now Application receives RTP events).

C's Connection/CallCtlConnection, TerminalConnection/CallCtlTerminalConnection will go into the Connected/Established/Active/Talking state.

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

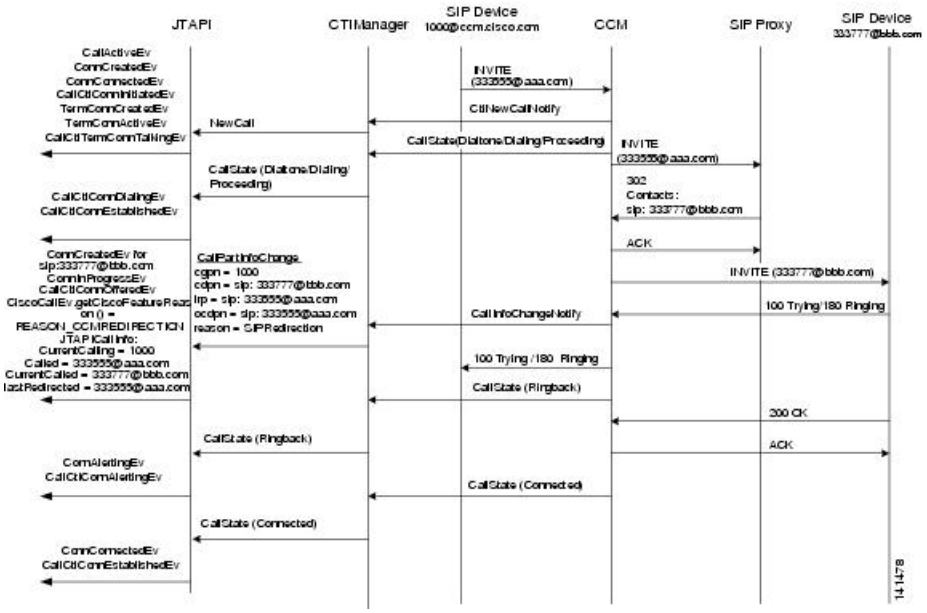
getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

SIP 3XX Redirection

3XX Redirection – 302 Moved Temporarily



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and Connection and CallCtlConnection events for 1000

JTAPI reports CallCtlConnEstablishedEv

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302, the Cisco Unified Communications Manager initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com.

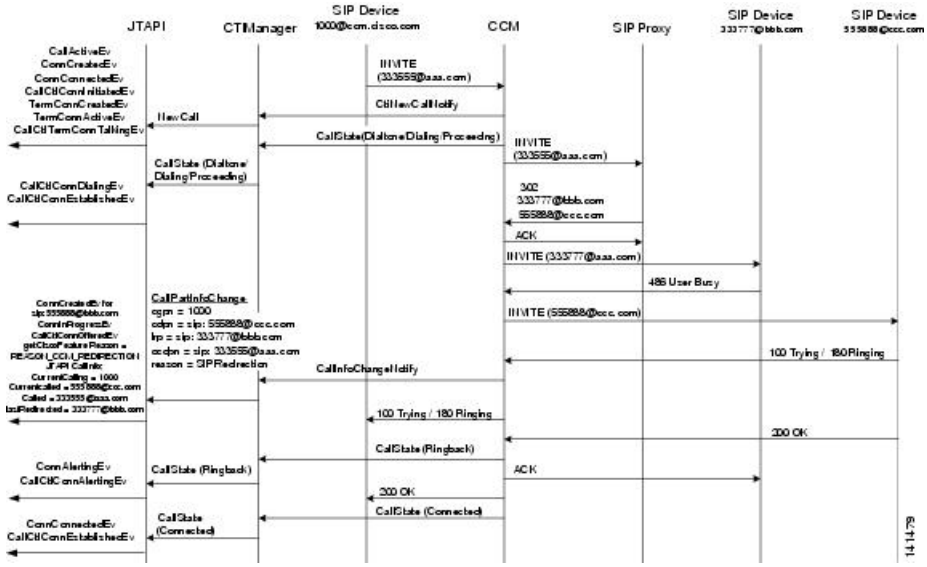
CallPartyInfoChange event is reported to application based on the SIPAlertInd from a Cisco Unified Communications Manager, if the called party information is changed.

JTAPI reports connection created events for 333777@bbb.com

CTI reports CtiCallStateNotify (Ringback) and CtiCallStateNotify (Connected).

JTAPI reports ConnAlertingEv and ConnEstablishedEv for far end.

3XX Redirection – Contact Busy



JTAPI CTI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and Connection and CallCtlConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Cisco Unified Communications Manager initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com.

A 486 user busy response is reported by 333777@bbb.com. Based on this response the Cisco Unified Communications Manager initiates a call to 555888@cisco.com.

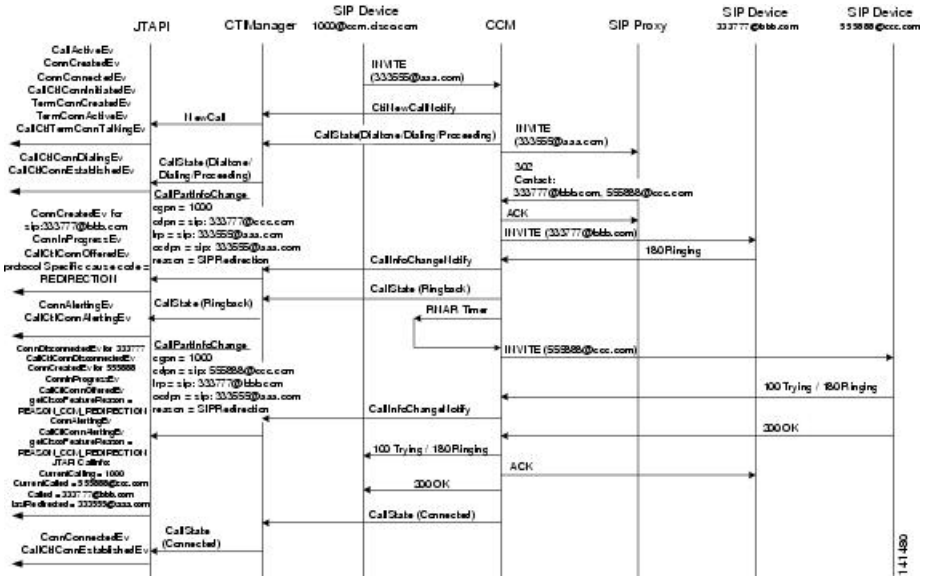
CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI reports connection created event for 555888@cisco.com.

CTI also reports CtiCallStateNotify (Ringback) and CtiCallStateNotify (Connected).

JTAPI reports CallCtlConnAlertingEv and CallCtlConnEstablishedEv for the new party

3XX Redirection – Contact Does Not Answer



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Cisco Unified Communications Manager initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com. The Cisco Unified Communications Manager starts the RNAR timer.

CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI reports connection created events for 333777

RNAR timer expires and based on this expiration the Cisco Unified Communications Manager initiates a call to 555888@cisco.com.

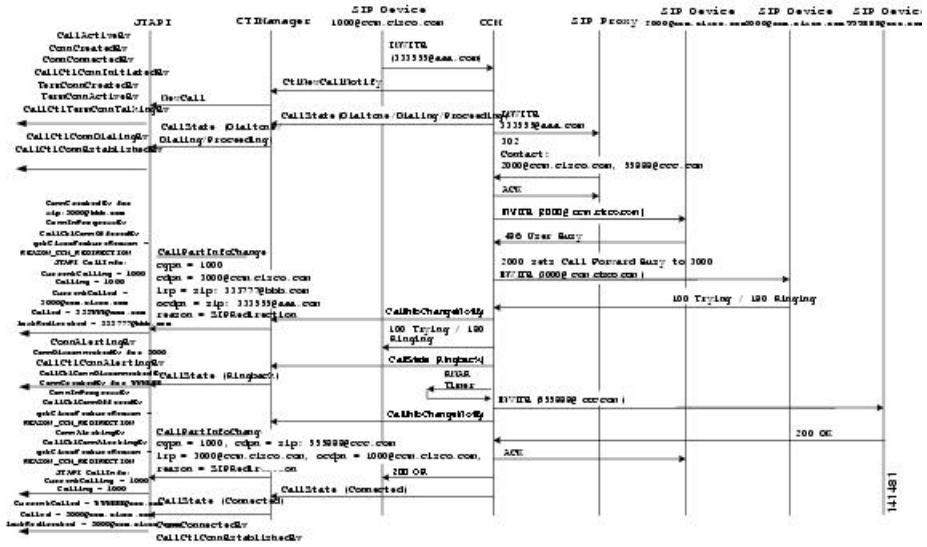
CallPartyInfoChange event is reported to application based on the SIPAlertInd/CcNotifyReq from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI removes connection for 333777 and creates connection for 555888

CTI also reports CtiCallStateNotify (Connected).

JTAPI reports CallCtlConnEstablishedEv for 555888

3XX Redirection – Contact Within Cisco Unified Communications Manager Cluster Configured with Call Forward



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Cisco Unified Communications Manager initiates a call to the first contact in the Target list based on the q value to 2000@ccm.cisco.com.

A 486 user busy response is reported by 2000@ccm.cisco.com. 2000 has Call Forward busy configured so the Cisco Unified Communications Manager initiates a call to 3000@ccm.cisco.com. The Cisco Unified Communications Manager also starts the RNAR timer.

CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI reports connection created event for 3000

3000 does not answer and RNAR timer expires and based on this expiration the Cisco Unified Communications Manager initiates a call to 555888@cisco.com.

CallPartyInfoChange event is reported to application based on the SIPAlertInd/CcNotifyReq from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI destroys connection for 3000 and creates connection for 555888

CTI also reports CtiCallStateNotify (Connected).

JTAPI reports CallCtlConnEstablishedEv for 555888

3XX Redirection – Non-Available Target Member

JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. 302 contains target list of 1212@ccm.cisco.com and 2000@ccm.cisco.com. 1212@ccm.cisco.com is an invalid DN. The Cisco Unified Communications Manager tries to contact 1212@ccm.cisco.com first, but gets an invalid DN and so attempts to place the call to 2000@ccm.cisco.com.

CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI reports connection created event for 2000

CTI also reports CtiCallStateNotify (Ringback/Connected).

JTAPI reports CallCtlConnAlertingEv and CallCtlConnEstablishedEv for 2000.

SIP Support

S.No	Scenario	Events
1	External SIP phone(external@someserver.com) calls A, A is monitored by application. Assuming external sip phone uses uri and not DN.	Event delivered to call observer on A CallActiveEv ConnCreatedEv A Conn CreatedEv unknown getCurrentCallingPartyInfo().geUrlInfo().getUser() returns external. getCurrentCallingPartyInfo().geUrlInfo().getHost() returns someserver.com getCurrentCallingPartyInfo().geUrlInfo().getUrlType() returns SIP_URL_TYPE
2	7970 runs SIP protocol with 2 max calls set. 3rd call comes in with GCID = GCID3	GCID3 CallActiveEv GCID3 ConnCreatedEv A GCID3 ConnFailedEv A GCID3 callInvalidEv
3	7960 running SIP is included in the control list. Applications add callobserver on the terminal	Exception is thrown to addobserver exception. TerminalRestrictedEv will be delivered if the status changed.

SIP Trunk Early Offer

Scenario One

Early offer call on a IPV4 mode. CTIPort or RP supports this feature.Application opens provider and adds address, terminal and call observers. (Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to the other address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT

Action	Result	Call information
Application sets the RTP parameters(IPv4 address and port) Application answers the call on B.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false

Scenario Two

Early offer call on a IPv4 mode. CTIPort or RP supports this feature. Application does not set RTP parameters in time(Fail Call Over SIP Trunk if MTP Allocation Fails = true).

Application opens provider and adds Address, terminal and call observers.(Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTermina.register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT for registration type and CiscoTerminal.IP_ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application does not sets the RTP parameters (IPv4 address and port).	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv TermA Gc1 ConnFailedEv A Gc1 CallCtlConnFailedEv A Gc1 CallInvalidEv PlatformException: Could not meet post condition of connect()	ev.getCause() = CAUSE_RESOURCES_NOT_AVAILABLE

Scenario Three

Early offer call on a IPv4 mode. CTIPort or RP supports this feature. Application does not setPort in time. (Fail Call Over SIP Trunk if MTP Allocation Fails = false)

Application opens provider and adds address, terminal and call observers. (Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTermina.register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application does not sets the RTP parameters. (IPv4 address and port) B answers the call. Application sets the RTP parameters.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv	isRTPRequired() = true.

Scenario Four

Early offer call on a dynamically registered IPv6 only CtiPort/RP with DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv6 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B	
App answers the call on B Application sets RTP parameters.	GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false

Scenario Five

Early Offer call on a dynamically registered CtiPort or RP with DYNAMIC_MEDIA_REGISTRATION for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new BaseMediaTerminal.register() API and DYNAMIC_MEDIA_REGISTRATION for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B.	
App answers the call on B Application sets RTP parameters	GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = true

Scenario Six

Two applications registering same CTIPort or RP with different values for registrationType.(dynamic).

Action	Result	Call information
Application1 registers the terminal TermA dynamically using the new CiscoBaseMediaTerminal .register() API and DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application2 registers the terminal TermA dynamically usig the new CiscoBaseMediaTerminal .register() API and passes something other than DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	PlatformException:	CTIERR_MEDIA_ALREADY_ TERMINATED_DYNAMIC_ GETPORT_SUPPORT
Application3 registers the terminal TermA dynamically usig the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Scenario Seven

Application sets RTP parameters again for an early offer call with dynamically registered terminal having DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application sets the RTP parameters (IPv4 address and port). Application answers the call on B. Application sets the RTP parameters again.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA InvalidStateException GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false. CTIERR_OPERATION_NOT_ AVAILABLE_IN_CURRENT_STATE.

Scenario Eight

Transfer involving a early offer call

Application registers two terminals TermA and TermB dynamically with DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and for both. A call is established between TermA and TermB.

Action	Result	Call information
A puts the call on hold	GC1 CallCtlTermConnHeldEv TermA	
A initiates a call to C	GC2 CallActiveEv GC2 ConnCreatedEv A GC2 ConnConnectedEv A GC2 CallCtlConnInitiatedEv A GC2 TermConnCreatedEv TermA GC2 TermConnActiveEvTerm A GC2 CallCtlTermConnTalkingEv TermA GC2 CallCtlConnDialingEv A GC2 CallCtlConnEstablishedEv A GC2 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application sets the RTP parameters for TermA and opens the port(IPv4 address and port) App answers the call on C	GC2 TermConnRingingEv C GC2 CallCtlTermConnRingingEvImpl C GC2 CiscoMediaOpenLogicalChannelEv TermA GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv C CiscoRTPInputStartedEvs CiscoRTPOutputStartedEvs GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAletingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv	Ev.isRTPRequired() = false

Action	Result	Call information
A tansfers the two calls GC1.transfer(GC2)	GC2 ConnDisconnectedEv C Gc2 CallCtlConnDisconnectedEv C GC2 ConnDisconnectedEv A GC2 CallCtlConDisconnectedEv A GC2 CallInvalidEv GC1 ConnCreatedEv C GC1 CiscoMediaOpenLogicalChannelEv TermB	Ev.isRTPRequired() = true
Application sets the RTP parameters for TermB	GC1 ConnEstablishedEv C CiscoRTPInputStartedEv CiscoRTPOutputStartedEvs	

Scenario Nine

Hold Resume Scenario

The application registers terminal TermA with DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.

A call is established between TermA and TermB.

Action	Result	Call information
TermA puts the call on hold.	Gc1 CallCtlTermConnHeldEv CiscoRTPInputStopped EvCiscoRTPOutoutStoppedEv	
TermA resumes the call.	GC1 CiscoMediaOpenLogicalChannelEv TermA	Ev.isRTPRequired() = true
Application sets the RTP parameters for TermA.	CiscoRTPInputstartedEv CiscoRTPOutputStarted EvCallCtlTermConnTalkingEv	

Scenario Ten

Call from a terminal registered with registration type as CiscoBaseMediaTerminal .STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT.

Action	Result	Call information
Application registers the terminal statically using the new CiscoBaseMediaTerminal .register() API and passes STATIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEvterm ACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes STATIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT
Application sets the RTP parameters.(IPv4 address and port). Application answers the call on B.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	

Scenario Eleven

Two Applications registering same CTIPort or RP with different values for registrationType (static).

Action	Result	Call information
Application1 registers the terminal TermA statically using the new register() API and STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application2 registers the terminal TermA statically usig the new register() API and passes something other than STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	PlatformException	CTIERR_MEDIA_ALREADY_ TERMINATED_STATIC_ GETPORT_SUPPORT
Application3 registers the terminal TermA statically usig the new register() API and passes STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

SRTP Key Material

If this feature is enabled, it is expected to degrade the performance of Cisco Unified JTAPI. Performance degradation is because of encrypted signaling between CTI and JTAPI and also because of encrypted media between end points.

Scenario One

Action	Event
App adds CallObserver on an Address 1 and initiates a call to address2 and involves in secure media conversation. If user is authorized, then CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv contain key material.	CiscoRTPInputKeyEv CiscoRTPInputStartedEv CiscoRTPOutputKeyEv CiscoRTPOutputStartedEv

Scenario Two

Action	Event
Application adds TerminalObserver by enabling snapshotEnabled filter. Device is already in a secure call and queries invokes CiscoTerminal.createSnapshot ()	CiscoTermSnapshotEv using which applications can query getCiscoMediaCallSecurity () to find out if a call is secured or not.

Scenario Three

Action	Response
Application does not have a TLS link and tries to register with secure media.CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm)	PrivilegeViolationException is thrown to the application
Application has a secure media and registers CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm)	Request is successful

Super Provider Message Flow

The application tries to create Terminal for CTIPort1 that has Addresses 2000 and 2001. The following events get sent to the application.

No.	Action	Event
1	Application invokes CiscoProvider. CreateTerminal(CTIPort1) whereCiscoProviderCapabilities. canObserveAnyTerminal() returns TRUE.	JTAPI would return CiscoTerminal object and the following events get sent: CiscoTermCreatedEv CTIPort1<---------- CiscoAddrCreated 2000<---------- CiscoAddrCreated 2001<----------
2	If the application already has a terminal where the 2001 address already exists, that is, 2001 is a SharedLine Address. Now, the application invokes CiscoProvider.CreateTerminal(CTIPort1)	JTAPI would return CiscoTerminal object and the following events get sent CiscoTermCreatedEv CTIPort1<---------- CiscoAddrCreated 2000<---------- CiscoAddrAddedToTerminalEv 2001<--------
3	Application invokes CiscoProvider.CreateTerminal(CTIPortX)where CTIPortX does not exist in Cisco Unified Communications Manager cluster.	JTAPI would throw an exception: InvalidArgumentException
4	Application invokes CiscoProvider.CreateTerminal(CTIPort1) whereCiscoProviderCapabilities. canObserveAnyTerminal() returns FALSE.	JTAPI would throw an exception: PrivilegeViolationException

SuperProvider and Change Notification Enhancements Use Cases

New events have been added to JTAPI, which will be sent to applications in order to handle new failover scenario and change notification. This enhances JTAPI to handle failover scenarios and the time required to shift between Superprovider and normal user modes.

Scenario One

Superprovider user opens provider and opens a few devices in Superprovider mode which are not in control list. From admin pages, Superprovider privilege is removed.

Application receives CiscoProviderCapabilityChangedEvent event. JTAPI sends CiscoTermRemovedEv all the devices which are opened / acquired and are not in the control list. JTAPI will send provider OOS to application, CiscoTermRemovedEv to devices not in control list and will reopen connection to CTI. When connect succeeds, JTAPI will send provider in service event to the application. Else, it will close the provider.

Scenario Two

Normal user opens provider and opens a few devices in control list. From admin pages, Superprovider privilege is added to the user.

Application receives CiscoProviderCapabilityChangedEvent event. User will now be able to acquire/open devices not in its control list.

Scenario Three

Normal user opens provider and opens a few park DNs. From admin pages, park DN monitor privilege is removed for the user.

Application receives CiscoProviderCapabilityChangedEvent event. JTAPI will cleanup all park DN addresses.

Scenario Four

Normal user opens provider. From admin pages, park DN monitor privilege is added for the user.

Application receives CiscoProviderCapabilityChangedEvent event. Application registers the park DN monitoring feature and is able to monitor park DN.

Scenario Five

Normal user opens provider. From admin pages, “modify calling party” privilege is removed for the user.

Application receives CiscoProviderCapabilityChangedEvent event. Application is not able to change the calling party number during redirect. JTAPI will throw error if application tries to do this.

Scenario Six

Normal user opens provider. From admin pages, “modify calling party” privilege is added for the user.

Application receives CiscoProviderCapabilityChangedEvent event. Application is able to change the calling party number in a call during redirect.

Scenario Seven

Superprovider user opens provider and acquires a device not in control list. From admin pages, the device is deleted.

Application receives CiscoTermRemovedEv event. Device is closed from JTAPI perspective.

Support for Cisco Unified IP Phone 6901

Scenario 1

Phone A is a Cisco Unified IP Phone 6901 and phone B is a normal SCCP/SIP phone. Application is observing the devices A and B. Phone A is off-hook and application initiates a call through createCall() API from phone A to phone B.

Configuration:

- Phone A – Cisco Unified IP Phone 6901
- Phone B – SCCP/SIP Device

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	Application observes A and B.
A goes off-hook Application calls createCall() and call connect() API to Call B.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] GC1: TermConnDroppedEv [Term A] CallCtlTermConnDroppedEv [Term A] ConnDisconnectedEv A CallCtlConnDisconnectedEv A CallInvalidEv	

Action	Result	Call info
	GC2: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] CallCtlConnDialingEv A CallCtlConnEstablishedEv A ConnCreatedEv – B - ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv [Term B] TermConnRingingEv [Term B] CallCtlTermConnRingingEvImpl [Term B]	
B answers the call and A & B are connected.	GC2: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 2

Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application is observing both the devices A & B. User initiates a call from phone A to phone B. Phone B goes off-hook and answers the incoming call.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
B answers the call and A & B are connected.	GC2: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 3

Phone A is Cisco Unified IP Phone 6901 and phone B is a normal SCCP/SIP phone. Application is observing both the devices A and B. Phone A is on-hook and application initiates a call from phone A to phone B.

Configuration:

- Phone A – Cisco Unified IP Phone 6901
- Phone B – SCCP/SIP Device

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A is on-hook and application call createcall() and connect API to call B	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 4

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B is on-hook and application tries to answer the call on B.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A intitiates a call to B from application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B is on-hook and tries to answer the call from the application.	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 5

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Scenario 6

Call Park Reversion Timer is set to 30 seconds at service parameter page. Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call. B goes on-hook.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Action	Result	Call info
B goes on-hook. Call Park reversion Timer expires after 30 seconds and the call comes back to B. B tries to answer the call from the application.	GC1: ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv [Term B] TermConnRingingEv [Term B] CallCtlTermConnRingingEvImpl [Term B] Jtapi throwsEception: InvalidStateException.	Operation not available in current state.
B goes off-hook and answers the call.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 7

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application. C is off-hook and unparks the call from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Action	Result	Call info
C unparks the call from the application.	GC2: CallActiveEv ConnCreatedEv –C ConnConnectedEv – C CallCtlConnInitiatedEv - C - ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnOfferedEv ParkDN	
	GC1: ConnDisconnectedEv ParkDN CallCtlConnDisconnectedEv ParkDN	
	GC1: ConnCreatedEv –C ConnConnectedEv – C CallCtlConnEstablishedEv – C TermConnCreatedEv [Term C] TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C] ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnOfferedEv ParkDN	
	GC2: ConnDisconnectedEv ParkDN CallCtlConnDisconnectedEv ParkDN TermConnDroppedEv [Term C] CallCtlTermConnDroppedEv [Term C] ConnDisconnectedEv C GC2 CiscoCallChangedEv CallCtlConnDisconnectedEv C CallInvalidEv	

Scenario 8

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application. C is on-hook and unparks the call from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Action	Result	Call info
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL
C unparks the call from the application.	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 9

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B transfers the call to C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B makes consult call to C	GC1: CallCtlTermConnHeldEv B GC2: CallActiveEv ConnCreatedEv –B ConnConnectedEv – B CallCtlConnInitiatedEv - BTermConnCreatedEv - [Term B] TermConnActiveEv –[Term B] CallCtlTermConnTalkingEv –[Term B] -	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL

Action	Result	Call info
C goes off-hook and answers the call. B and C are connected.	GC2: ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL
B completes transfer by invoking GC1.transfer(GC2)	GC1: CiscoTermConnSelectChangedEv [Term B] GC2: CiscoTermConnSelectChangedEv [Term B] GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv	
	GC1: ConnCreatedEv - C ConnConnectedEv – C CallCtlConnEstablishedEv - C TermConnCreatedEv - [Term C] TermConnActiveEv –[Term C] CallCtlTermConnTalkingEv –[Term C]	
	GC2: TermConnDroppedEv [Term C] CallCtlTermConnDroppedEv [Term C] ConnDisconnectedEv C CallCtlConnDisconnectedEv C	

Action	Result	Call info
	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B GC2: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B CallInvalidEv CiscoTransferEndEv	

Scenario 10

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B does a conference with C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv – C	

Action	Result	Call info
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B makes consult call to C. C goes off-hook and answers the call.	GC1: CallCtlTermConnHeldEv B GC2: CallActiveEv ConnCreatedEv –B ConnConnectedEv – B - ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL

Action	Result	Call info
B conferences two calls by invoking GC1.conference(GC2)	GC1: CiscoTermConnSelectChangedEv [Term B] GC2: CiscoTermConnSelectChangedEv [Term B] GC1 CiscoConferenceStartEv GC2 CiscoCallChangedEv	
	GC1: ConnCreatedEv - C ConnConnectedEv – C CallCtlConnEstablishedEv - C TermConnCreatedEv - [Term C] TermConnActiveEv –[Term C] CallCtlTermConnTalkingEv –[Term C] GC2: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B CallInvalidEv GC1 CiscoConferenceEndedEv	

Scenario 11

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901 models. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B redirects the call to C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Aleta Device

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B redirects the call to C from the application.	GC1: ConnCreatedEv C ConnInProgressEv C CallCtlConnOfferedEv C TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B	

Action	Result	Call info
C is off-hook and answers the call from application.	CallCtlConnDisconnectedEv B ConnAlertingEv C CallCtlConnAlertingEv C TermConnCreatedEv [Term C] TermConnRingingEv [Term C] CallCtlTermConnRingingEvImpl [Term C] ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = A currentCalled = C CAUSE = CAUSE_NORMAL

Scenario 12

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing the corresponding button from the phone. B resumes the call by pressing Line key from the phone.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Aleta Phone

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

Action	Result	Call info
B answers the call, and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses Hold button from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses Line button from the phone and resumes the call.	GC1: CallCtlTermConnTalkingEv B	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 13

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing the button and goes on-hook. Now B tries to resume the call from application.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A initiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses hold hardkey from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	
B goes on-hook. B tries to resume the call from the application.	Exception: InvalidStateException.	Operation not available in current state.

Scenario 14

Application is observing devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing hard key and goes on-hook. Now B tries to resume the call by pressing the line hard key from phone.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses hold hardkey from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	
B goes on-hook. B tries to resume the call by pressing the line key from phone.	GC1: CallCtlConnTalkingEv [Term B] TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B TermConnDroppedEv [Term A] CallCtlTermConnDroppedEv [Term A] ConnDisconnectedEv A CallCtlConnDisconnectedEv A CallInvalidEv	Operation not available in current state.

Scenario 15

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. CFA is set to C at phone B. Application initiates a call from phone A to phone B. Due to CFA set to C, call goes to C. C goes off-hook and answers the call.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A intitiates a call to B from the application. B has CFA set to C. Call goes to C.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	CAUSE = CAUSE_NORMAL REASON = FORWARDALL
C goes off-hook and answers the call. A & C are connected.	GC1: ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 16

Application is observing the devices A and B. Phone C is not observed. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B redirects the call to C. C goes off-hook and answers the call.

Configuration:

- Phone A – SCCP/SIP Device

- Phone B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B. Phone C is not observed.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B	
A initiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B redirect the call to C. C goes off-hook and answers the call.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv C ConnAlertingEv C CallCtlConnAlertingEv C ConnConnectedEv C CallCtlConnEstablishedEv C	

Scenario 17

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B is Cisco Unified IP Phone 6901 and phone C is a normal SCCP/SIP phone. B and C have a shared line. Application initiates a call from phone A to shared line on B and C. B goes off-hook and answers the call.

Configuration:

- Phone A and C – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A initiates a call to the shared line on B and C	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B goes off-hook and answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnCreatedEv [Term C] TermConnPassiveEv [Term C] CallCtlTermConnInUseEv [Term C] TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	

SHA Support for Digital Signatures

The following tables display the CallInfo messages for the following three use cases:

- SHA-1 is the configured encryption algorithm

- SHA-512 is the configured encryption algorithm (Cisco JTAPI is version 11.5)
- SHA-512 is the configured encryption algorithm (Cisco JTAPI is a pre-11.5 version)

Table 352: SHA-1 is Configured

Action	CallInfo
TFTP File Signature Algorithm enterprise parameteris set to **SHA1** (the default value). JTAPIProperties.setSecurityPropertyForInstance() is invoked with CAPF login and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=true

Table 353: SHA-512 is Configured (Cisco JTAPI is version 11.5)

Action	CallInfo
TFTP File Signature Algorithm enterprise parameter is set to **SHA 512**. JTAPIProperties.setSecurityPropertyForInstance() is invoked with username and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=true

Table 354: SHA-512 is Configured (Cisco JTAPI is pre-11.5 version)

Action	CallInfo
TFTP File Signature Algorithm enterprise parameter is set to **SHA 512**. JTAPIProperties.setSecurityPropertyForInstance() is invoked with username and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=false

TLS Security

Message flow for updating certificate and establishing TLS certificate is illustrated in the following two figures.

Figure 22: CTI/API Security Approach

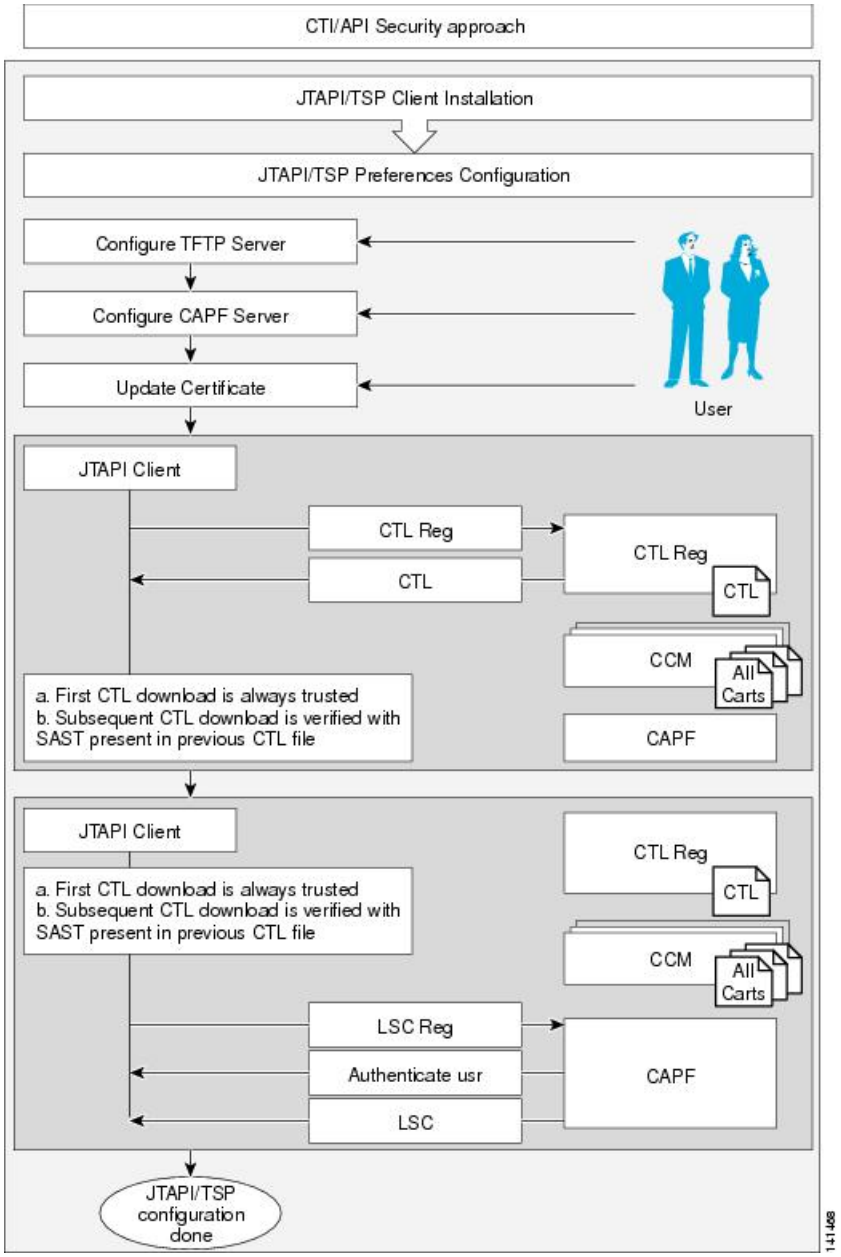
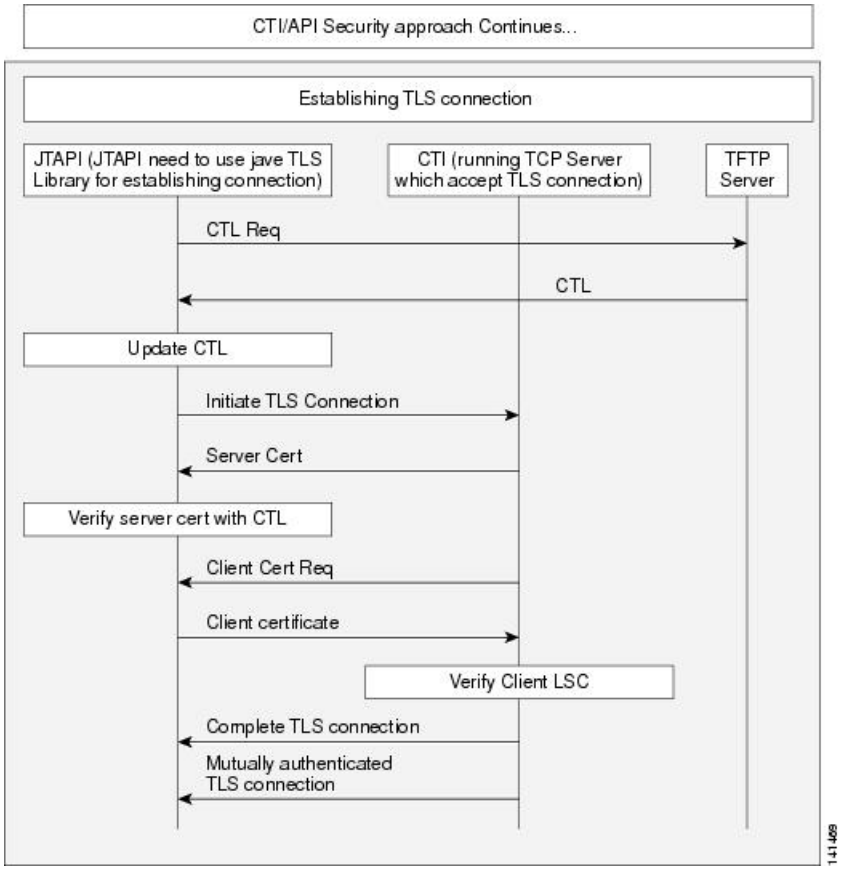


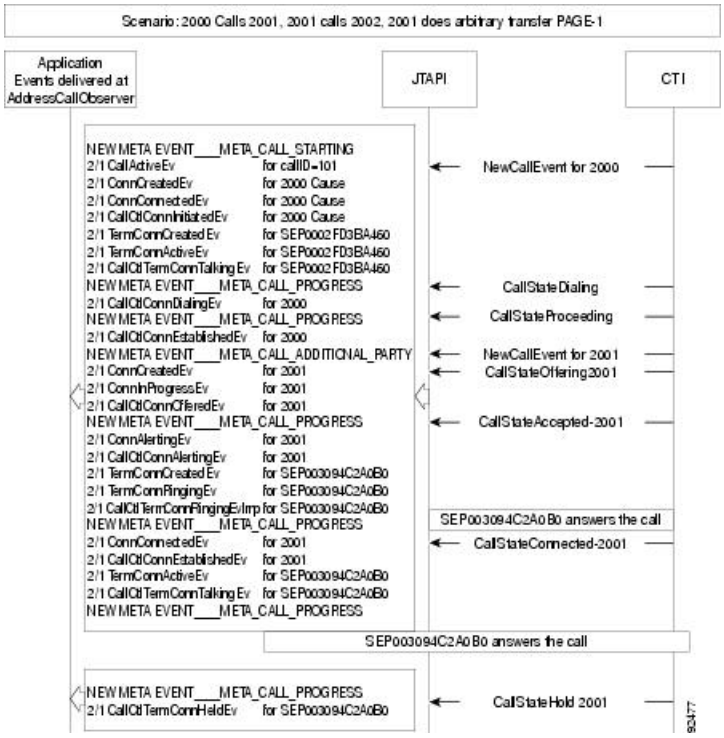
Figure 23: CTI/API Security Approach (Continued)



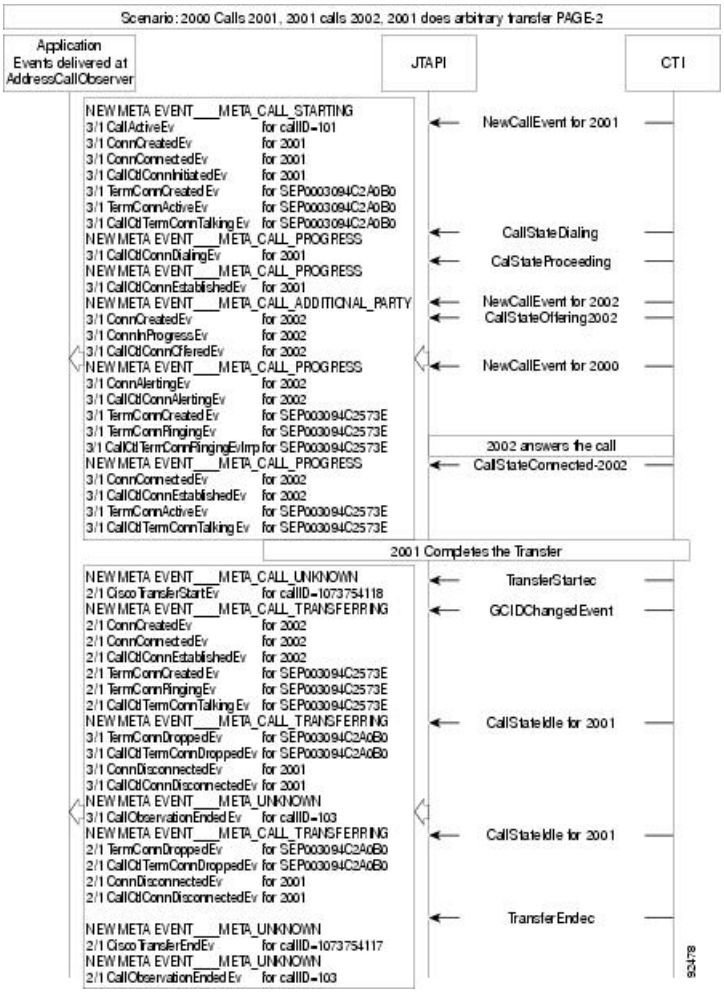
Transfer and Direct Transfer

The following diagrams illustrate the message flows for Transfer and Direct Transfer.

DirectTransfer/Arbitrary Transfer Scenario



Direct Transfer/Arbitrary Transfer-Page 2



Consult Transfer

The message flow for Consult Transfer acts the same as the flow for Arbitrary Transfer.

Unicode Support

Unicode Display Name Scenario

Scenario	Events Delivered to JTAPI Applications
A line is configured on IP phone A with no ASCII name and a Unicode name in Japanese. IP phone B is configured with ASCII name and no Unicode name. A calls B. Only B is observed.	Call info should contain: getCurrentCalledPartyDisplayName = asciiNameB getCurrentCalledPartyUnicodeDisplayName = null getCurrentCallingPartyDisplayName = null getCurrentCallingPartyUnicodeDisplayName = japaneseNameA
A, B and C are in conference.	DisplayName does not apply. Applications should consider “conference” as the called party.
Shared lines – A and B are shared lines with different locales. A calls C. C is unobserved.	Calling party Unicode display name can change between A and B.

GetLocale and UniCodeCapabilities of Terminal

Scenario	Events delivered to JTAPI applications
A line is configured on IP phone A with no ASCII name and Unicode name in Japanese. Application adds TerminalObserver on the Device. Application queries the following using CiscoTerminal.	CiscoTerminalInServiceEv contains getLocale = JAPANESE getSupportedEncoding = UCS2UNICODE_ENCODING CiscoTerminal.getLocale = JAPANESE CiscoTerminal. getSupportedEncoding = UCS2UNICODE_ENCODING

Unrestricted Unified CM

Use Case One

The application tries to register with an insecure CTI Port to the unrestricted Cisco Unified Communications Manager.

Action	Result	Call information
Application opens a provider with the unrestricted Cisco Unified Communications Manager and tries to register with an insecure phone CTI Port 'A'.	[Term A] CiscoTermOutOfService[A] CiscoAddrOutOfServiceEv[Term A] CiscoTermInServiceEv[A] CiscoAddrInServiceEv	

Variance

Application tries to register with an insecure route point to the unrestricted Cisco Unified Communications Manager.

Use Case Two

Restricted Cisco Unified Communications Manager is upgraded to unrestricted Cisco Unified Communications Manager. The application tries to register with a secure phone after the upgrade.

In some of the scenarios, where the application registers a device in a secure mode, the registeration is successful but eventually can be rejected with a new error code - CiscoTermRegisterationFailedEv.

Variance

Application tries to register a secure Route Point after an upgrade from a restricted Cisco Unified Communications Manager to unrestricted Cisco Unified Communications Manager.

Video Capabilities and Multi-Media Information

The following sections describe use cases that are related to video capabilities and multi-media information feature.

Scenario One

Phone A is video capable, telepresence capable, with 1 screen and a camera, and in registered state. User1 has phone A in the control list. User queries for multimedia capabilities before adding a terminal observer.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getVid eoCapability() on termA		termA. getCiscoMulti MediaCapabilityInfo() .getVideoCa pability() = CiscoMultiMedia CapabilityInfo.ENABLED
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getTel epresenceInfo () on termA		termA. getCiscoMulti MediaCapabilityInfo() .getTelepres enceInfo () = CiscoMultiMedia CapabilityInfo.TELEPRESENC EINTEROP_ENABLED
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getScr eenCount () on termA		termA. getCiscoMulti MediaCapabilityInfo() .getScreenC ount () = 1

Scenario Two

Phone A is a video disabled SIP Phone (In Cisco Unified CM Administration Phone page, Video Capabilities field is “Disabled”). Phone A is in a registered state.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA.getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Enabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. ENABLED
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. ENABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Disabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. DISABLED

Scenario Three

Phone A is a video disabled SCCP Phone (In Cisco Unified CM Administration Phone page, Video Capabilities field is “Disabled”). Phone A is in a registered state.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED

Action	Events	Call Info
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Enabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. ENABLED
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. ENABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Disabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv will not be delivered to applications, as the device will unregister and register back. In this case applications can query video capability after the device is registered.	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED

Scenario Four

Phone A is video capable, telepresence capable, has 1 screen, has a camera, and is in an unregistered state. User1 has phone A in the control list. User queries for multimedia capabilities before adding a terminal observer.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getVideoCapability() on termA		InvalidStateException: Terminal is not in registered state.
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getTelepresenceInfo() on termA		InvalidStateException: Terminal is not in registered state.
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getScreenCount() on termA		InvalidStateException: Terminal is not in registered state.

Scenario Five

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 has phone A in the control list. Application queries for mutlimedia capabilities on CiscoTerminal.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	termA. getCiscoMulti MediaCapabilityInfo().getVideoCapability() = NONE
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo.ENABLED
User1 invokes CiscoTerminal. i getCiscoMulti MediaCapabilityInfo(). getTelepresenceInfo() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getTelepresenceInfo() = CiscoMultiMediaCapabilityInfo.TELEPRESENCEINTEROP_ENABLED
User1 invokes CiscoTerminal. i getCiscoMulti MediaCapabilityInfo(). getScreenCount() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getScreenCount () = 1

Scenario Six

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registers it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Action	Events	Call Info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario Seven

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registers it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario Eight

Basic Video call: Phone A is video enabled, Telepresence Enabled and has 1 screen. Phone B has video disabled, Telepresence Disabled and has 0 screens. Both phones are in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscoTermInServiceEv TermA CiscoTermInServiceEv TermB	

Action	Events	Call Info
User1 adds call observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEvImpl B	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresence capable device(CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getCallingTerminal VideoCapability() on GC1.		The API returns 0, indicating video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getTelepresenceInfo() on GC1.		The API returns 0, indicating telepresence capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 01 screen, for TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresence capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getCallingTerminal VideoCapability() on GC1.		The API returns 0, indicating not video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getTelepresenceInfo() on GC1.		The API returns 0, indicating device is not telepresence capable (TELEPRESENCEINTEROP_DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 0 screens, for TermB

Scenario Nine

Shared Line: Phone A has video enabled, Phone B has video disabled and Phone B' has video enabled. B' is in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone B'	CiscotermInServiceEv TermB'	
User1 adds call observes on the address B'	CiscoAddrInServiceEv B'	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo().getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. DISABLED) for TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo().getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		Application will receive the following incorrect data: The API returns 1, indicating not video capable device(CiscoMultiMediaCapabilityInfo. ENABLED) for Term B.

Scenario Ten

Shared Line: Phone A has video enabled, Phone B has video disabled and Phone B' has video enabled. B and B' is in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone B and Phone B'	CiscotermInServiceEv TermB CiscotermInServiceEv TermB'	
User1 adds call observes on the address B and B'	CiscoAddrInServiceEv B CiscoAddrInServiceEv B'	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.).getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo. .getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. DISABLED) for TermB'
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.).getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.).getCallingTerminalVideoCapability() on GC1.		Application will receive the following incorrect data: The API returns 1, indicating not video capable device(CiscoMultiMediaCapabilityInfo. ENABLED) for Term B.

Scenario Eleven

Basic Video call: Phone A is video enabled, Telepresence enabled and has 1 screen. Phone B has video disabled, Telepresence disabled and has 0 screens. Phone A is in User1's control list, Phone A is observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns -1, indicating video capability is not known (UKNOWN) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns -1, indicating telepresence capability is not known (UKNOWN) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns -1, indicating screen count capability is not known TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns -1, indicating video capability is not known (UKNOWN) for TermB

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns -1, indicating telepresence capability is not known (UKNOWN) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns -1, indicating screen count capability is not known TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresenc capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ ENABLED) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for Term B.

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns 0, indicating device is not telepresence capable (TELEPRESENCEINTEROP_ DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 0 screens, for TermB

Scenario Twelve

MultiMedia Streams: Phone A and B have video enabled, and both phones are in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionAddress() on Terminal A		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionPort() on Terminal A		The API returns port number from which media will be directed.

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionAddress() on Terminal A		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionPort() on Terminal A		The API returns the payload format.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getPayloadType() on Terminal A		The API returns the maximum bit rate.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getMaxBitRate() on Terminal A		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal A		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaType() on Terminal A		
CiscoMultiMediaStreamsInfoEv. getProperties(). isKeyInfoPresent() on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal A		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal A		The API returns 3(MEDIA_ENCRYPTED_KEYS _UNAVAILABLE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionAddress() on Terminal B		The API returns IP address to which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionPort() on Terminal B		The API returns port number to which media will be directed.

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionAddress() on Terminal B		The API returns IP address from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionPort() on Terminal B		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getPayloadType() on Terminal B		The API returns the payload format.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getMaxBitRate() on Terminal B		The API returns the maximum bit rate.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaType() on Terminal B		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties(). isKeyInfoPresent() on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal B		The API returns 3(MEDIA_ENCRYPTED_KEYS _UNAVAILABLE)
B holds the call.		
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal A		The API returns 3(INACTIVE)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 3(INACTIVE)
B unholds the call.		
CiscoMultiMediaStreamsInfoEv. getCallingTerminalVideoCapability() on GC1.		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)

Scenario Thirteen

Redirect: Phone A, B, and C have video enabled, and A, B phones are in User1's control list, and Phone C is in User2's control list.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B User2 adds terminal observers on Phone C.	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC	
User1 adds callObserves on the address A and B User2 adds callObserves on the address C	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal A		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties().getMultiMediaType() on Terminal A		The API returns 2(MAIN_VIDEO)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties().isKeyInfoPresent() on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties().getMultiMediaType() on Terminal B		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties().isKeyInfoPresent() on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal B		The API returns 2(MAIN_VIDEO) 3(MEDIA_ENCRYPTED _KEYS_UNAVAILABLE)
B redirects the call to C. C answers		
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal B		The API returns 3(INACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal C		The API returns 0(ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal A		The API returns 0(ACTIVE)

Action	Events	Call Info
App does CiscoCall. getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capability is enabled (CiscoMultiMediaCapabilityInfo.ENABLED) for TermA (far-end party).
App does CiscoCall. getCalledTerminalMult iMediaCapabilityInfo() on GC2.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).
CiscoMultiMediaStreamsInfoEv. getCalledTerminalMult iMediaCapabilityInfo on GC2.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).

Scenario Fourteen

Transfer: Phone A, B and C have video enabled, and A, B phones are in User1's control list and C is in User2's control list, A, B and C are in cluster1.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B User2 adds terminal observers on Phone C.	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC	
User1 adds callObserves on the address A and B User2 adds callObserves on the address C	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal A		The API returns 0 (TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal A		The API returns 2 (MAIN_VIDEO)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties ().isKeyInfoPresent () on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal B		The API returns 0 (TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal B		The API returns 2 (MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties ().isKeyInfoPresent () on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaEncryptionKeyInfo () on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaSecurityIndicator () on Terminal B		The API returns 3 (MEDIA_ENCRYPTED_ KEYS_UNAVAILABLE)
B does a consult call to C. C answers		
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal B		The API returns 0 (ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal C		The API returns 0 (ACTIVE)
App does CiscoCall. getCalledTerminalMultiMediaCapabilityInfo () on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB (far-end party).

Action	Events	Call Info
B does GC1.transfer (GC2). C answers		
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal B		The API returns 3 (INACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal C		The API returns 0 (ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal A		The API returns 0 (ACTIVE)
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfoon GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo () on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).
App does CiscoCall. getCalledTerminalMultiMediaCapabilityInfo ().getCallingTerminalVideoCapability () onGC1.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo () on GC1.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).

Scenario Fifteen

Negotiated video capability will be reported to the called party across a cluster call (over SIP – ICT trunk) using early offer (Phone A is a video disabled SIP Phone in cluster 1 and Phone B is a video enabled SIP Phone in cluster 2). User1 has Phone A in the control list, Phone A is observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	

Action	Events	Call Info
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermA.

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB.
Variation 1: A and B are SIP Phone and have video enabled. User1 makes a call from A to B. B answers the call. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device(CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.

Scenario Sixteen

Multiple redirects over SIP trunk (A, B, C and D are video enabled SIP Phones, A is in cluster 1 and B, C, D are in cluster 2). Phone A, B, C and D are in User1's control list, Phone A, B, C and D are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A B, C, and D	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC CiscotermInServiceEv TermD	
User1 adds callObserves on the address A, B, C, and D	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C CiscoAddrInServiceEv D	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.
App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB.

Action	Events	Call Info
B redirects the call to C.		
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
C redirects the call to D.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device(CiscoMultiMedia CapabilityInfo.ENABLED) for TermD.

Scenario Seventeen

Redirect over SIP trunk (Phone A is video enabled SIP Phone, and Phone B and C have video disabled. Phone A is in cluster 1 and Phone B and C are in cluster 2). Phone A and B are in User1's control list and Phone C is in User2's control list, Phone A, B and C are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermA.
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo. DISABLED) for TermC.

Action	Events	Call Info
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo. ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo. ENABLED) for TermC.

Scenario Eighteen

Shared Line – Hold and Resume scenario over SIP trunk (Phone A and C are video enabled SIP Phones and Phone B is video disabled, Phone A is in cluster 1 and Phone B and C are in cluster 2. Phone B and C are shared lines). Phone A and B are in User1's control list, Phone A and B are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
B redirects the call to C. C answers		
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo() .getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo() .getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.

Action	Events	Call Info
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo() .getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo() .getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermC.

Scenario Nineteen

Multiple redirects over H323 ICT trunk (A, B, C and D are video enabled SIP Phones, A is in cluster 1 and B, C, D are in cluster 2). Phone A, B, C and D are in User1's control list, Phone A, B, C and D are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A B, C, and D	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC CiscotermInServiceEv TermD	
User1 adds call Observes on the address A B, C, and D	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C CiscoAddrInServiceEv D	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermB. The API returns -1, indicating screen count isUNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermB.
B redirects the call to C. C answers		

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.
C redirects the call to D.		

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermD. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.

Scenario Twenty

Redirect over H323 trunk (Phone A is video enabled SIP Phone and Phone B and C have video disabled, Phone A is in cluster 1 and Phone B and C are in cluster 2). Phone A and B are in User1's control list and Phone C is in user2's control list, Phone A, B and C are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
B redirects the call to C. C answers		
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.

Action	Events	Call Info
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermC.
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
C redirects the call to D.		

Action	Events	Call Info
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo. getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermD. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.

Video On Hold

Pre-conditions to all video on hold use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Device A (IP Phone – Name: “SEP2401C7824EA3”, Line A1 (dn: 9000))
- Device B (IP Phone – Name: “SEP2401C7824EAE”, Line B1 (dn: 9001))
- The content id corresponding to VoH stream is contentID1.
- User1 has in its control list: Devices A and B. All devices and lines are observed.

Scenario One

Invoke hold() on basic call between two ip phones to stream video to the held party

Action	Events	Call information / Notes
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9001").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE	
Device B answers the call.	GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE	
User1 invokes hold("contentID1") on terminal connection of Device A.	GC1: CallCtlTermConnHeldEv SEP2401C7824EA3 **Note** You are able to see video streamed to the device that is placed on hold.	

Action	Events	Call information / Notes
A disconnects the call.	GC1: TermConnDroppedEv SEP2401C7824EA3 GC1: CallCtlTermConnDroppedEv SEP2401C7824EA3 GC1: ConnDisconnectedEv 9000 GC1: CallCtlConnDisconnectedEv 9000 GC1: TermConnDroppedEv SEP2401C7824EAE GC1: CallCtlTermConnDroppedEv SEP2401C7824EAE GC1: ConnDisconnectedEv 9001 GC1: CallCtlConnDisconnectedEv 9001 GC1: CallInvalidEv GC1: CallObservationEndedEv	

Verification Involving PSTN Reachability

Scenario 1

A calls B in other cluster (Normal Call); application is observing A

Action	Result	Call info
A dials B, B rings.	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	

Scenario 2

A calls B in other cluster (VIPR Call - Call gets routed through IP trunk); application is observing A

Action	Result	Call info
A dials B, B rings	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	

Scenario 3

A calls B, within same cluster. B redirects the call to external party C, the redirected call goes through IP Trunk due to VIPR feature; application is observing both A and B.

Action	Result	Call info
A calls B withis same cluster	GC1: CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEv TB	

Action	Result	Call info
B redirects the call to external party C	GC1: TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv C ConnConnectedEv C CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	CiscoFeatureReason = CiscoFeatureReason.REASON_REDIRCT CallCtlCause = CallCtlEv.CAUSE_REDIRECTED CiscoFeatureReason.REASON_REDIRECT

Scenario 4

A calls external party B; call goes through IP trunk but later call quality degrades and VIPR PSTN Fallback happens; application is observing A.

Action	Result	Call info
A dials B(VIPR Call), B rings	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	
B answers	TA: CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	

Action	Result	Call info
Call Quality Degrades and call falls back to PSTN network	TA: CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	

Scenario 5

A calls B, B transfers the call to external Party C; Transferred call goes through IP trunk due to VIPR feature; application is observing both A and B.

Action	Result	Call info
A calls B; B answers	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B	

Action	Result	Call info
B makes a consult call to external party C.	GC2: ConsultCallActiveEv GC2: ConnCreatedEv B GC2: ConnConnectedEv B GC2: CallCtlConnInitiatedEv B GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB GC2: CallCtlConnEstablishedEv B GC2: ConnCreatedEv C GC2: ConnConnectedEv for C GC2: CallCtlConnNetworkReachedEv GC2: CallCtlConnNetworkAlertingEv	
B completes the transfer; Two calls are transferred; A and C get connected over IP trunk due to VIPR feature	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoTransferStartedEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1: ConnConnectedEv for C GC2 ConnDisconnectedEv for C GC2 CallCtlConnDisconnectedEv C GC1 TermConnDroppedEv B GC1 CallCtlTermConnDroppedEv B GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectecEv B GC2 TermConnDroppedEv B GC2 CallCtlTermConnDroppedEv B GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectecEv B GC2 CallInvalidEv GC1 CiscoTransferEndEv	CiscoFeatureReason = CiscoFeatureReason. REASON_TRANSFEREDCALL

Scenario 6

A calls B, within the same cluster. B has CFA set to Forward all the calls to external party C, the forwarded call goes through IP Trunk due to VIPR feature. Application is observing A.

Action	Result	Call info
A calls B within the same cluster	GC1: CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A	
Call at B gets forwarded to external party C	GC1: CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv ConnCreatedEv C ConnConnectedEv C CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	CiscoFeatureReason = CiscoFeatureReason.REASON_FORWARDALL CallCtlCause = CallCtlEv.CAUSE_REDIRECTED

Whisper Coaching

Use Case One - Monitoring with Mode as WHISPER

Start and Stop Whisper monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (CI1). B calls start monitor using GC2. The application has a call observer on both A and B. The monitoring capability enabled.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X… GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA in GC1 and mode as Whisper	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.WHISPER_MONITORGC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TBCiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A
A puts the call with X on hold	GC1: CallCtlTermConnHeldEv TA GC2:CiscoRTPInputStoppedEv TB GC2: CiscoRTPOutputStoppedEv TB GC1: CiscoRTPOutputStoppedEv TA GC1: CiscoRTPInputStoppedEv TA	

Action	Events	Call information
A resumes the call	GC1: CallCtlTermConnTalking TA GC1: CiscoRTPOutputStartedEv TA GC1: CiscoRTPInputStartedEv TA GC2: CiscoRTPOutputStartedEv TB GC2:CiscoRTPInputStartedEv TB	
B calls drop() on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv TB GC2: TermConnDroppedEv TB GC2: ConnDisconnectedEvB GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Two - Snapshot Use Case for Whisper Monitoring

- A is monitor target. B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Another application adds call observer on A after monitoring session is established.

Action	Events	Call information
	CallActiveEv for callID = GC1 Cause: CAUSE_SNAPSHOT GC1:ConnCreatedEv for A Cause: CAUSE_SNAPSHOT GC1:ConnConnectedEv for A Cause: CAUSE_SNAPSHOT GC1: ConnConnectedEv X GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_SNAPSHOT GC1: CiscoTermConnMonitoringStartEv TA Cause: CAUSE_SNAPSHOT getMonitorType() = CiscoCall.WHISPER_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_SNAPSHOT address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

- A is monitor target and B is monitor initiator. Caller X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Another application adds call observer on B after monitoring sessions are established.

Action	Events	Call information
	GC2:CallActive Cause: CAUSE_SNAPSHOT	GC1:
	GC2: ConnCreatedEv for B Cause: : CAUSE_SNAPSHOT	Calling: B
	GC2: ConnConnectedEv B	Called: A
	GC2: CallCtlConnEstablishedEv for B Cause: : CAUSE_SNAPSHOT	LRP: null
	GC2: TermConnCreatedEv TB	Current calling: B
	GC2: TermConnActiveEv TB	Current called:A
	GC2: CallCtlTermConnTalkingEv TB B Cause: : CAUSE_SNAPSHOT	
	GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET	
	GC2: ConnConnectedEv A	
	GC2: CallCtlConnEstablishedEv A	
	GC2:CiscoTermConnMonitorTargetInfoEv TB	
	Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1	
	CiscoMonitorInitatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
	[Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.]	

Use Case Three

Start Silent Monitoring then update the monitorType to Whisper mode followed by redirect and the updateMonitorType back to Silent monitor.

Whisper monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2(mode = silent). Application has call observer on both A and B. Application has monitoring capability enabled. App updates the monitor mode to Whisper. B redirects the monitoring call to observed party C. App updates the monitor mode to silent and later drops monitoring call to stop monitoring.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL	GC1:
	GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL	Calling: X
	GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL	Called: A
	GC1: ConnConnectedEv X	LRP: null
	GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL	Current calling: X
	GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	Current called:A
	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as Silent	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.SILENT_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.SILENT_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.SILENT_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A
The application calls updateMonitorType() API on TermConn of B with mode as Whisper.	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv TB getMonitorType() = CiscoCall.WHISPER_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStartedEv TB	

Action	Events	Call information
B redirects the call to C and C answers.	GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnOfferedEv C GC2: TermConnCreatedEv TC GC2: TermConnActiveEv TC GC2: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv TB GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnectedEv TB GC2: CallCtlConnDisconnectedEv TB GC2: CiscoTermConnMonitorTargetInfoEv TC CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR GC2; CiscoTermConnMonitorInitatorInfoEv TA Cause: CAUSE_NORMAL address:C, device name: TC CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
The application calls updateMonitorType() API on TermConn of C with mode as Silent	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv TC getMonitorType() = CiscoCall.SILENT_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStoppedEv TC	
C calls drop on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv TC GC2: TermConnDroppedEv TC GC2: ConnDisconnectedEv C GC2: CallCtlConnDisconnectedEv C GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Four

Secured Whisper Monitoring followed by redirect to non-secured supervisor

Whisper monitor: A is monitor target, B is monitor initiator. Both A and B are Encrypted devices. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2(mode = whisper). Application has call observer on both A and B. Application has monitoring capability enabled. B redirects to observed party C which is non-secured.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as whisper	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.WHISPER_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
B redirects the call to C and C answers	GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnOfferedEv C GC2: TermConnCreatedEv TC GC2: TermConnActiveEv TC GC2: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv TB GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnectedEv TB GC2: CallCtlConnDisconnectedEv TB GC2: CiscoTermConnMonitorTargetInfoEv TC Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC2: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:C, device name: TC CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR GC2: ConnFailedEv C GC2: CallCtlConnFailedEv C cause : CAUSE_BCNAUTHORISED GC2: CallInvalidEv GC2: CiscoAddrMonitorTerminatedEv B cause : CAUSE_BCNAUTHORISED	

Use Case Five

Agent greeting is played and the application initiates the monitoring request (Silent/Whisper).

JTAPI throws InvalidStateException with error "CTIERR_RESOURCE_NOT_AVAILABLE" when the application sends a monitor request and when an agent greeting is played at that time.

Use Case Six - Tone Direction Interaction Use Cases

1. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE

- If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE

2. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
3. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
4. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
5. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
6. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
7. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
8. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
9. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
10. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
11. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY

- If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
- If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY

12. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
13. Service Parameter = PLAYTONE_BOTHLOCALANDREMOTE; API Request (startMonitor/updateMonitorType) = Any tone
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY

Use Case Seven

Supervisor B is a shared line and supervisor updates monitorType from silent to whisper. Address B is a shared line configured on terminal T1(initiator) and T2.

Whisper monitor: A is monitor target, B(T1) is monitor initiator. X calls A, A answers the call GC1 (ci1). B(T1) calls start monitor using GC2(mode = silent). Application has call observer on all A, B(T1) and B(T2_. Application has monitoring capability enabled. App updates the monitor type to Whisper and later drops monitoring call to stop monitoring.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as Silent	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv B(T1) GC2: TermConnActiveEv B(T1) GC2: TermConnCreatedEv for B(T2) GC2: TermConnPassiveEv for B(T2) GC2: CallCtlTermConnTalkingEv B(T1) B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv B(T1) getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv B(T1) Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.SILENT_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TAgetMonitorType() = CiscoCall.SILENT_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: B(T1) CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.SILENT_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
The application calls updateMonitorType() API on TermConn of B with mode as Whisper	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv B(T1) getMonitorType() = CiscoCall.WHISPER_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStartedEv B(T1)	
B calls drop on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv BT(1) GC2: TermConnDroppedEv B(T1) GC2: CallCtlTermConnDroppedEv BT(2) GC2: TermConnDroppedEv B(T2) GC2: ConnDisconnectedEv B GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Eight

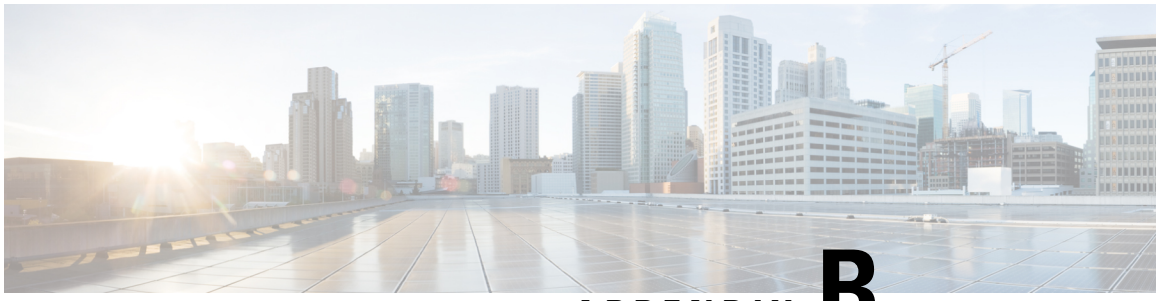
Agent is shared line. After monitoring Agent holds and its shared line resumes.

A(T1) is monitor target and shares a line with Terminal T2 , B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on all A(T1), A(T2) and B. Application has monitoring capability enabled.

Action	Events	Call information
A(T1) answers GC1B calls start monitor using GC2 giving CI1, A and TermA(T1) in GC1 and mode as Whisper		GC1: Calling: X Called: A LRP: null Current calling: X Current called:A GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv T1 CiscoRTPInputStartedEv T1 GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv T1(A) getMonitorType() = CiscoCall.WHISPER_MONITOR	

Action	Events	Call information
	GC1: CiscoTermConnMonitorInitiatorInfoEv T1(A) Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
A(T1) puts the call on hold	GC1: CallCtlTermConnHeldEv A(T1) GC1: CallCtlTermConnActiveEv A(T2) GC1: CallCtlTermConnHeldEv A(T2) GC2:CiscoRTPInputStoppedEv TB GC2: CiscoRTPOutputStoppedEv TB GC1: CiscoRTPOutputStoppedEv A(T1) GC1: CiscoRTPInputStoppedEv A(T1)	
A(T2) resumes the call	GC1: CallCtlTermConnTalkingEv A(T2) GC1: CallCtlTermConnPassiveEv A(T1) GC1: CiscoRTPOutputStartedEv A(T2) GC1: CiscoRTPInputStartedEv A(T2) GC2: CallCtlTermConnDroppedEv TB GC2: TermConnDroppedEv TB GC2: ConnDisconnectedEvB GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	



APPENDIX B

Cisco Unified JTAPI Classes and Interfaces

This appendix contains a listing of all classes and interfaces that are available in the Cisco Unified JTAPI implementation:

- Cisco Unified JTAPI Version 1.2 Classes and Interfaces, on page 1599, which lists all the JTAPI v 1.2 classes and methods. The supported classes and methods have a check mark in the Cisco Unified JTAPI Support column.
- Cisco Unified JTAPI Extension Classes and Interfaces, on page 1617, which lists the Cisco Unified JTAPI extension classes and methods.
- Cisco Trace Logging Classes and Interfaces, on page 1622, which lists the error tracing classes and methods.

Cisco Unified JTAPI Version 1.2 Classes and Interfaces

This section includes the following:

• Media Events Package, on page 1616
• Unsupported Packages, on page 1616

Core Package

The following table lists each JTAPI interface in the JTAPI Core Package followed by the associated method (s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnified JTAPI support	Comments
Address	addCallObserver	Yes	
	addressObserver	Yes	
	getAddressCapabilities	Yes	
	getCallObservers	Yes	
	getCapabilities	Yes	
	getConnections	Yes	
	getName	Yes	
	getObservers	Yes	
	getProvider	Yes	
	getTerminals	Yes	
	removeCallObserver	Yes	
	removeObserver	Yes	
AddressObserver	addressChangedEvent	Yes	
Call	addObserver	Yes	
	connect	Yes	A CallObserver must exist for the Terminal or Address originating the call. The FeaturePriority parameter is not supported.
	getCallCapabilities	Yes	
	getCapabilities	Yes	
	getConnections	Yes	

Class names	Method names	CiscoUnified JTAPI support	Comments
	getObservers	Yes	
	getProvider	Yes	
	getState	Yes	
	removeObserver	Yes	
CallObserver	callChangedEvent	Yes	
Connection	disconnect	Yes	
	getAddress	Yes	
	getCall	Yes	
	getCapabilities	Yes	
	getConnectionCapabilities	Yes	
	getState	Yes	
	getTerminalConnections	Yes	
JtapiPeer	getName	Yes	
	getProvider	Yes	
	getServices	Yes	
JtapiPeerFactory	getJtapiPeer	Yes	
Provider	addObserver	Yes	
	createCall	Yes	
	getAddress	Yes	
	getAddressCapabilities ()	Yes	
	getAddressCapabilities (Terminal)	Yes	
	getAddresses	Yes	
	getCallCapabilities ()	Yes	
	getCallCapabilities (Terminal, Address)	Yes	

Class names	Method names	CiscoUnified JTAPI support	Comments
	getCalls	Yes	This method returns calls only when there are CallObservers attached to Addresses or Terminals, when a RouteAddress is registered for routing, or when a CiscoMediaTerminal is registered.
	getCapabilities	Yes	
	getConnectionCapabilities ()	Yes	
	getConnectionCapabilities (Terminal, Address)	Yes	
	getName	Yes	
	getObservers	Yes	
	getProviderCapabilities ()	Yes	
	getProviderCapabilities (Terminal)	Yes	
	getState	Yes	
	getTerminal	Yes	
	getTerminalCapabilities ()	Yes	
	getTerminalCapabilities (Terminal)	Yes	
	getTerminalConnectionCapabilities ()	Yes	
	getTerminalConnectionCapab ilities (Terminal)	Yes	
	getTerminals	Yes	
	removeObserver	Yes	
	shutdown	Yes	
ProviderObserver	providerChangedEvent	Yes	
Terminal	addCallObserver	Yes	
	addObserver	Yes	
	getAddresses	Yes	
	getCallObservers	Yes	

Class names	Method names	CiscoUnified JTAPI support	Comments
	getCapabilities	Yes	
	getName	Yes	
	getObservers	Yes	
	getProvider	Yes	
	getTerminalCapabilities	Yes	
	getTerminalConnections	Yes	
	removeCallObserver	Yes	
	removeObserver	Yes	
TerminalConnection	answer	Yes	
	getCapabilities	Yes	
	getConnection	Yes	
	getState	Yes	
	getTerminal	Yes	
	getTerminalConnectionCapabilities	Yes	
TerminalObserver	terminalChangedEvent	Yes	

Call Center Package

The following table lists each JTAPI interface in the JTAPI Call Center Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support
ACDAddress	getACDManagerAddress	
	getLoggedOnAgents	
	getNumberQueued	
	getOldestCallQueued	
	getQueueWaitTime	
	getRelativeQueueLoad	
ACDAddressObserver		

Class names	Method names	CiscoUnifiedJTAPI support
ACDConnection	getACDManagerConnection	
ACDManagerAddress	getACDAddresses	
ACDManagerConnection	getACDConnections	
Agent	getACDAddress	
	getAgentAddress	
	getAgentID	
	getAgentTerminal	
	getState	
	setState	
AgentTerminal	addAgent	
	getAgents	
	removeAgents	
	setAgents	
AgentTerminalObserver		
CallCenterAddress	addCallObserver	
CallCenterCall	connectPredictive	
	getApplicationData	
	getTrunks	
	setApplicationData	
CallCenterCallObserver		
CallCenterProvider	getACDAddresses	
	getACDManagerAddresses	
	getRouteableAddresses	
CallCenterTrunk	getCall	
	getName	
	getState	
	getType	
RouteAddress	cancelRouteCallback	Yes

Class names	Method names	CiscoUnifiedJTAPI support
	getActiveRouteSessions	Yes
	getRouteCallback	Yes
	registerRouteCallback	Yes
RouteCallback	reRouteEvent	Yes
	routeCallbackEndedEvent	Yes
	routeEndEvent	Yes
	routeEvent	Yes
	routeUsedEvent	Yes
RouteSession	endRoute	Yes
	getCause	Yes
	getRouteAddress	Yes
	getState	Yes
	selectRoute	Yes

Call Center Capabilities Package

The following table lists each JTAPI interface in the JTAPI Call Center Capabilities Package followed by the associated method(s), and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support
ACDAddressCapabilities	canGetACDManagerAddress	
	canGetLoggedOnAgents	
	canGetNumberQueued	
	canGetOldestCallQueued	
	canGetQueueWaitTime	
	canGetRelativeQueueLoad	
ACDConnectionCapabilities	canGetACDManagerConnection	
ACDManagerAddressCapabilities	canGetACDAddresses	
ACDManagerConnectionCapabilities	canGetACDConnections	
AgentTerminalCapabilities	canHandleAgents	

Class names	Method names	CiscoUnifiedJTAPI support
CallCenterAddressCapabilities	canAddCallObserver	
CallCenterCallCapabilities	canConnectPredictive	
	canGetTrunks	
	canHandleApplicationData	
CallCenterProviderCapabilities	canGetACDAddresses	Yes
	canGetACDManagerAddresses	Yes
	canGetRouteableAddresses	Yes
RouteAddressCapabilities	canRouteCalls	Yes

Call Center Events Package

The following table lists each JTAPI interface in the JTAPI Call Center Events Package followed by the associated method(s), and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support
ACDAddrBusyEv		
ACDAddrEv	getAgent	
	getAgentAddress	
	getAgentTerminal	
	getState	
	getTrunks	
ACDAddrLoggedOffEv		
ACDAddrLoggedOnEv		
ACDAddrNotReadyEv		
ACDAddrReadyEv		
ACDAddrUnknownEv		
ACDAddrWorkNotReadyEv		
ACDAddrWorkReadyEv		
AgentTermBusyEv		
AgentTermEv	getACDAddress	

Class names	Method names	CiscoUnifiedJTAPI support
	getAgent	
	getAgentAddress	
	getAgentID	
	getState	
AgentTermLoggedOffEv		
AgentTermLoggedOnEv		
AgentTermNotReadyEv		
AgentTermReadyEv		
AgentTermUnknownEv		
AgentTermWorkNotReadyEv		
AgentTermWorkReadyEv		
CallCentCallAppDataEv	getApplicationData	
CallCentCallEv	getCalledAddress	
	getCallingAddress	
	getCallingTerminal	
	getLastRedirectedAddress	
	getTrunks	
CallCentConnEv		
CallCentConnInProgressEv		
CallCentEv	getCallCenterCause	
CallCentTrunkEv	getTrunk	
CallCentTrunkInvalidEv		
CallCentTrunkValidEv		
ReRouteEvent		Yes
RouteCallbackEndedEvent	getRouteAddress	Yes
RouteEndEvent		Yes
RouteEvent	getCallingAddress	Yes
	getCallingTerminal	Yes

Class names	Method names	CiscoUnifiedJTAPI support
	getCurrentRouteAddress	Yes
	getRouteSelectAlgorithm	Yes
	getSetupInformation	Yes
RouteSessionEvent	getRouteSession	Yes
RouteUsedEvent	getCallingAddress	Yes
	getCallingTerminal	Yes
	getDomain	Yes
	getRouteUsed	Yes

Call Control Package

The following table lists each JTAPI interface in the JTAPI Call Control Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI Implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
CallControlAddress	cancelForwarding	Yes	Only for Call Forward All
	getDoNotDisturb		
	getForwarding	Yes	Only for Call Forward All
	getMessageWaiting		
	setDoNotDisturb		
	setForwarding	Yes	Only for Call Forward All
	setMessageWaiting		
CallControlCall	addParty		
	conference	Yes	In a consult conference scenario, only OriginalCall.conference (ConsultCall) is supported. ConsultCall.conference (OriginalCall) is not supported.
	consult(TerminalConnection)	Yes	
	consult(TerminalConnection, String)	Yes	
	drop	Yes	

Class names	Method names	CiscoUnifiedJTAPI support	Comments
	getCalledAddress	Yes	
	getCallingAddress	Yes	
	getCallingTerminal	Yes	
	getConferenceController	Yes	
	getConferenceEnable	Yes	
	getLastRedirectedAddress	Yes	
	getTransferController	Yes	
	getTransferEnable	Yes	
	offHook	Yes	
	setConferenceController	Yes	
	setConferenceEnable	Yes	
	setTransferController	Yes	
	setTransferEnable	Yes	
	transfer(Call)	Yes	In a consult transfer scenario, only OriginalCall.transfer (ConsultCall) is supported. ConsultCall.transfer (OriginalCall) is not supported.
	transfer(String)	Yes	
CallControlCallObserver		Yes	
CallControlConnection	accept	Yes	
	addToAddress	Yes	
	getCallControlState	Yes	
	park	Yes	
	redirect	Yes	Redirect allows a connection in the CallControlConnection. ESTABLISHED state to be redirected.
	reject	Yes	
CallControlForwarding	getDestinationAddress		
	getFilter		

Class names	Method names	CiscoUnifiedJTAPI support	Comments
	getSpecificCaller		
	getType		
CallControlTerminal	getDoNotDisturb		
	pickup (Address, Address)		
	pickup (Connection, Address)		
	pickup (TerminalConnection, Address)		
	pickupFromGroup(Address)		
	pickupFromGroup(String, Address)		
	setDoNotDisturb		
CallControlTerminalConnection	getCallControlState	Yes	
	hold	Yes	
	join	Yes	Only implemented for CiscoIntercomAddresses
	leave		
	unhold	Yes	
CallControlTerminalObserver			

Call Control Capabilities Package

The following table lists each JTAPI interface in the JTAPI Call Control Capabilities Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support
CallControlAddressCapabilities	canCancelForwarding	Yes
	canGetDoNotDisturb	Yes
	canGetForwarding	Yes
	canGetMessageWaiting	Yes
	canSetDoNotDisturb	Yes
	canSetForwarding	Yes

Class names	Method names	CiscoUnifiedJTAPI support
	canSetMessageWaiting	Yes
CallControlCallCapabilities	canAddParty	Yes
	canConference	Yes
	canConsult	Yes
	canConsult(TerminalConnection)	Yes
	canConsult(TerminalConnection, String)	Yes
	canDrop	Yes
	canOffHook	Yes
	canSetConferenceController	Yes
	canSetConferenceEnable	Yes
	canSetTransferController	Yes
	canSetTransferEnable	Yes
	canTransfer	Yes
	canTransfer(Call)	Yes
	canTransfer(String)	Yes
CallControlConnectionCapabilities	canAccept	Yes
	canAddToAddress	Yes
	canPark	Yes
	canRedirect	Yes
	canReject	Yes
CallControlTerminalCapabilities	canGetDoNotDisturb	Yes
	canPickup	Yes
	canPickup(Address, Address)	Yes
	canPickup(Connection, Address)	Yes
	canPickup(TerminalConnection, Address)	Yes
	canPickupFromGroup	Yes
	canPickupFromGroup(Address)	Yes
	canPickupFromGroup(String, Address)	Yes

Class names	Method names	CiscoUnifiedJTAPI support
	canSetDoNotDisturb	Yes
CallControlTerminalConnectionCapabilities	canHold	Yes
	canJoin	Yes
	canLeave	Yes
	canUnhold	Yes

Call Control Events Package

The following table lists each JTAPI interface in the JTAPI Call Control Events Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
CallCtlAddrDoNotDisturbEv	getDoNotDisturbState		
CallCtlAddrEv			
CallCtlAddrForwardEv	getForwarding	Yes	
CallCtlAddrMessageWaitingEv	getMessageWaitingState		
CallCtlCallEv	getCalledState	Yes	
	getCallingAddress	Yes	
	getCallingTerminal	Yes	
	getLastRedirectedAddress	Yes	
CallCtlConnAlertingEv		Yes	
CallCtlConnDialingEv	getDigits	Yes	
CallCtlConnDisconnectedEv		Yes	
CallCtlConnEstablishedEv		Yes	
CallCtlConnEv		Yes	
CallCtlConnFailedEv		Yes	
CallCtlConnInitiatedEv		Yes	
CallCtlConnNetworkAlertingEv		Yes	
CallCtlConnNetworkReachedEv		Yes	
CallCtlConnOfferedEv		Yes	

Class names	Method names	CiscoUnifiedJTAPI support	Comments
CallCtlConnQueuedEv	getNumberInQueue	Yes	
CallCtlConnUnknownEv		Yes	
CallCtlEv	getCallControlCause	Yes	
CallCtlTermConnBridgedEv			
CallCtlTermConnDroppedEv		Yes	
CallCtlTermConnEv		Yes	
CallCtlTermConnHeldEv		Yes	
CallCtlTermConnInUseEv			
CallCtlTermConnRingingEv		Yes	
CallCtlTermConnTalkingEv		Yes	
CallCtlTermConnUnknownEv		Yes	
CallCtlTermDoNotDisturbEv			
CallCtlTermEv			

Capabilities Package

The following table lists each JTAPI interface in the JTAPI Capabilities Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
AddressCapabilities	isObservable	Yes	
CallCapabilities	canConnect	Yes	
	isObservable	Yes	
ConnectionCapabilities	canDisconnect	Yes	
ProviderCapabilities	isObservable	Yes	
TerminalCapabilities	isObservable	Yes	
TerminalConnectionCapabilities	canAnswer	Yes	

Events Package

The following table lists each JTAPI interface in the JTAPI Events Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI Implementation.

Class names	Method names	CiscoUnifiedJTAPI support
AddrEv	getAddress	Yes
AddrObservationEndedEv		Yes
CallActiveEv		Yes
CallEv	getCall	Yes
CallInvalidEv		Yes
CallObservationEndedEv	getEndedObject	Yes
ConnAlertingEv		Yes
ConnConnectedEv		Yes
ConnCreatedEv		Yes
ConnDisconnectedEv		Yes
ConnEv	getConnection	Yes
ConnFailedEv		Yes
ConnInProgressEv		Yes
ConnUnknownEv		Yes
Ev	getCause	Yes
	getID	Yes
	getMetaCode	Yes
	getObserved	Yes
	isNewMetaEvent	Yes
ProvEv	getProvider	Yes
ProvInServiceEv		Yes
ProvObservationEndedEv		Yes
ProvOutOfServiceEv		Yes
ProvShutdownEv		Yes
TermConnActiveEv		Yes

Class names	Method names	CiscoUnifiedJTAPI support
TermConnCreatedEv		Yes
TermConnDroppedEv		Yes
TermConnEvgetTerminalConnection		Yes
TermConnPassiveEv		
TermConnRingingEv		Yes
TermConnUnknownEv		Yes
TermEv	getTerminal	Yes
TermObservationEndedEv		Yes

Media Package

The following table lists each JTAPI interface from the JTAPI Media Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
MediaCallObserver		Yes	
MediaTerminalConnection	generateDtmf	Yes	
	getMediaAvailability		
	getMediaState		
	setDtmfDetection	Yes	
	startPlaying		
	startRecording		
	stopPlaying		
	stopRecording		
	useDefaultMicrophone		
	useDefaultSpeaker		
	usePlayURL		
	useRecordURL		

Media Capabilities Package

The following table lists each JTAPI interface in the JTAPI Media Capabilities Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
MediaTerminalConnection Capabilities	canDetectDtmf	Yes	
	canGenerateDtmf	Yes	
	canStartPlaying	Yes	
	canStartRecording	Yes	
	canStopPlaying	Yes	
	canStopRecording	Yes	
	canUseDefaultMicrophone	Yes	
	canUseDefaultSpeaker	Yes	
	canUsePlayURL	Yes	
	canUseRecordURL	Yes	

Media Events Package

The following table lists each JTAPI interface in the JTAPI Media Events Package followed by the associated method(s) and whether the classes are supported by the Cisco Unified JTAPI implementation.

Class names	Method names	CiscoUnifiedJTAPI support	Comments
MediaEv	getMediaCause	Yes	
MediaTermConnAvailableEv			
MediaTermConnDtmfEv	getDtmfDigit	Yes	
MediaTermConnEv		Yes	
MediaTermConnStateEv	getMediaState		
MediaTermConnUnavailableEv			

Unsupported Packages

The following table shows the JTAPI packages that are not supported by the Cisco Unified JTAPI implementation.

Unsupported JTAPI packages
JTAPI Phone Package
JTAPI Phone Capabilities Package
JTAPI Phone Events Package
JTAPI Private Data Package
JTAPI Private Data Capabilities Package
JTAPI Private Data Events Package

Cisco Unified JTAPI Extension Classes and Interfaces

Cisco Unified JTAPI Extension Classes

Table 355: Cisco Unified JTAPI Extension Classes

Cisco extension classes	Method names
CiscoMediaCapability	getMaxFramesPerPacket() getPayloadType() toString()
CiscoG711MediaCapability	
CiscoG723MediaCapability	getBitRate() toString()
CiscoGSMMediaCapability	
RegistrationException	
UnregistrationException	

Cisco Unified JTAPI Extension Interfaces

Table 356: Cisco Unified JTAPI Extension Interfaces and Their Methods

Cisco extension interfaces	Method names
CiscoAddrCreatedEv	getAddress()
CiscoAddress	getType()

Cisco extension interfaces	Method names
CiscoAddressObserver	
CiscoAddrEv	
CiscoAddrInService	
CiscoAddrOutOfService	
CiscoCall	getCallID()
CiscoCallEv	
CiscoCallID	getCall() intValue()
CiscoConferenceEndEv	getConferenceCall() getFinalCall() getHeldConferenceController() getTalkingConferenceController()
CiscoConferenceStartEv	getConferenceCall() getFinalCall() getHeldConferenceController() getTalkingConferenceController()
CiscoConnection	getConnectionID() getReason() redirect(String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, String preferredOriginalCalledParty, String facCode, String cmcCode, int featurePriority, byte[] applicationXMLData)
CiscoConnectionID	getConnection() intValue()
CiscoConsultCall	getConsultingTerminalConnection()
CiscoConsultCallActiveEv	getHeldTerminalConnection()
CiscoEv	
CiscoJtapiPeer	

Cisco extension interfaces	Method names
CiscoMediaTerminal	getRTPInputProperties() getRTPOutputProperties() register(InetAddress, int) unregister()
CiscoProvEv	
CiscoProvider	getCallbackGuardEnabled() getMediaTerminal() getMediaTerminals() setCallbackGuardEnabled() getRemoteTerminals() getRemoteTerminal(String name)
CiscoProvConnToLeastPriorCtiServerEv	
CiscoProvFallbackToPrimNwCompltdEv	
CiscoProvPrimNwReachableEv	getReachableCtiServers()
CiscoProviderObserver	
CiscoProvTerminalRemoteDestinationChangedEv	getTerminal() getRemoteDestinations() isMyAppLastToSetActiveRD() getIPAddressingMode() getIPV4Address() getIPV6Address()
CiscoRecorderInfo	getRecordingType()
CiscoRemoteDestinationInfo	getRemoteDestinationName() getRemoteDestinationNumber() getIsActiveRD()

Cisco extension interfaces	Method names
CiscoRemoteTerminal	getAllRemoteDestinations() getActiveRemoteDestinations() setActiveRemoteDestination(String remoteDestinationNumber, boolean isActiveRD) addRemoteDestination(String remoteDestinationName, String remoteDestinationNumber, boolean isActiveRD) removeRemoteDestination(String remoteDestinationNumber) removeAllRemoteDestinations() updateRemoteDestinationName(String remoteDestinationNumber, String remoteDestinationName) updateRemoteDestinationNumber(String remoteDestinationNumber, StringnewRemoteDestinationNumber) updateRemoteDestination(String remoteDestinationNumber, String remoteDestinationName, String newRemoteDestinationNumber, boolean isActiveRD) isRegisteredByThisApp() Cisco Extend & Connect (CTI Remote Device) getRegistrationType() isMyAppLastToSetActiveRD()
CiscoRouteSession	getCall() selectRoute(String[] routeSelected, int callingSearchSpace, String[] modifyingCallingNumber, String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption, String[] facCode, String[] cmcCode, int featurePriority, byte[][] applicationXMLData)
CiscoRTPInputProperties	getBitRate() getEchoCancellation() getLocalAddress() getLocalPort() getPacketSize() getPayloadType()
CiscoRTPInputStartedEv	getRTPInputProperties()
CiscoRTPInputStoppedEv	

Cisco extension interfaces	Method names
CiscoRTPOutputProperties	getBitRate() getMaxFramesPerPacket() getPacketSize() getPayloadType() getPrecedenceValue() getRemoteAddress() getRemotePort()
CiscoRTPOutputStartedEv	getRTPOutputProperties()
CiscoRTPOutputStoppedEv	
CiscoSynronousObserver	
CiscoTermCreatedEv	getTerminal()
CiscoTermEv	
CiscoTerminal	getRegistrationState() register() unregister() getType() getTypeName()
CiscoTerminalConnection	startRecording(int playToneDirection, int invocationType) stopRecording(int invocationType)
CiscoTerminalObserver	
CiscoTermInServiceEv	
CiscoTermOutOfServiceEv	
CiscoTransferEndEv	getFinalCall() getTransferController() getTransferredCall()
CiscoTransferStartEv	getFinalCall() getTransferController() getTransferredCall()
ObjectContainer	getObject() setObject()

Cisco extension interfaces	Method names
RTPBitRate	
RTPPayload	

Cisco Trace Logging Classes and Interfaces

Cisco Trace Logging Classes

Table 357: Cisco Trace Logging Classes

Cisco Trace Logging class	Method names
LogFileOutputStream	close() flush() getCurrentFile() getFileExtension() getFileNameBase() getMaxFiles() getMaxFileSize() write(byte[], int, int) write(int)
NullTraceWriter	close() flush() getEnabled() print(String) println(String)
OutputStreamTraceWriter	close() flush() getEnabled() print(String) println(String) setOutputStream(OUputStream

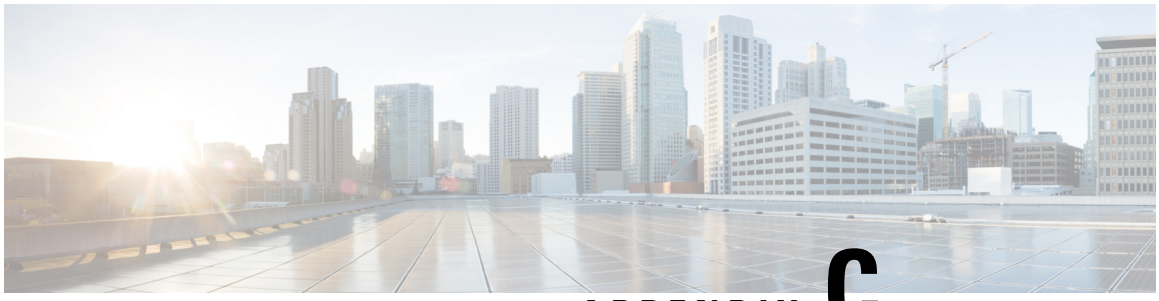
Cisco Trace Logging class	Method names
TraceManagerFactory	getModules() registerModule(String) registerModule(TraceModule) registerModule(TraceModule, OutputStream)

Cisco Trace Logging Interfaces

Table 358: Cisco Trace Logging Interfaces

Cisco Trace Logging interfaces	Method names
ConditionalTrace	disable() enable()
Trace	append(Object) append(String) getName() isEnabled() print(Object) print(String) print(String, Object) print(String, String) println(Object) println(String) println(String, Object) println(String, String) setDefaultMnemonic(String)

Cisco Trace Logging interfaces	Method names
TraceManager	disableAll() disableTimeStamp() enableAll() enableTimeStamp() getConditionalTrace(String) getConditionalTrace(String, String) getName() getOutputStream() getSubFacilities() getTraces() getTraceWriter() getUnconditionalTrace(String) getUnconditionalTrace(String, String) removeTrace(String) removeTrace(Trace) setOutputStream(OutputStream) setSubFacilities() setTraceWriter()
TraceModule	getTraceManager() getTraceModuleName()
TRACETYPE	
TraceWriter	close() flush() getEnabled() print(String) println(String)
UnconditionalTrace	



APPENDIX C

Troubleshooting Cisco Unified JTAPI

This appendix contains CTI Error Codes, CiscoEvent IDs, and other information to assist with troubleshooting efforts.

CTI Error Codes

Error code	Description
ASSOCIATED_LINE_NOT_OPEN	This error indicates that the request is issued on a line, which is not open
CALL_ALREADY_EXISTS	This error indicates that another call already exists on the line
CALL_DROPPED	The call dropped after the feature request (hold, unhold, transfer, or conference) but before the request was completed.
CALLHANDLE_NOTINCOMINGCALL	This error indicates that an attempt is made to answer a call that either does not exist or is not in the correct state
CALLHANDLE_UNKNOWN_TO_LINECONTROL	This error indicates that attempt to redirect call that was unknown to line control
CANNOT_OPEN_DEVICE	This error indicates that device open failed because the associated device is unregistering
CANNOT_TERMINATE_MEDIA_ON_PHONE	This error indicates that media cannot be terminated by an application when the device is a physical phone (the phone always terminates the media)
CFWDALL_ALREADY_SET	This error indicates that attempt to set CFWALL while it is already set

Error code	Description
CFWDALL_DESTN_INVALID	This error indicates that attempt to CFWALL to an invalid destination
CLUSTER_LINK_FAILURE	This error indicates that link to one of the cisco unified communications managers failed in the cluster (network error)
COMMAND_NOT_IMPLEMENTED_ON_DEVICE	This error indicates that device does not support the command.
CONFERENCE_ALREADY_PRESENT	This error indicates that attempt to conference a party that is already in conference
CONFERENCE_FAILED	This error indicates that conference completion was not successful.
CONFERENCE_FULL	This error indicates that all conference bridges are busy.
CONFERENCE_INACTIVE	This error indicates that attempt to complete conference while consult conference is not active
CONFERENCE_INVALID_PARTICIPANT	This error indicates that an attempt to conference to self or an invalid participant
CTIERR_ACCESS_TO_DEVICE_DENIED	This error indicates that the access to device is denied.
CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	This error indicates that the application softkeys are already controlled by another application
CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	This error indicates that application data size has exceeded limit
CTIERR_BIB_NOT_CONFIGURED	This error indicates built in bridge is not configured
CTIERR_BIB_RESOURCE_NOT_AVAILABLE	This error indicates that built in bridge resource not available
CTIERR_CALL_MANAGER_NOT_AVAILABLE	This error indicates that Communications Manager is not available currently
CTIERR_CALL_NOT_EXISTED	This error indicates that call does not exist
CTIERR_CALL_PARK_NO_DN	This error indicates no call park DN
CTIERR_CALL_REQUEST_ALREADY_OUTSTANDING	This error indicates call request already outstanding
CTIERR_CALL_UNPARK_FAILED	This error indicates that call unpark did not succeed
CTIERR_CAPABILITIES_DO_NOT_MATCH	This error indicates that capabilities do not match
CTIERR_CLOSE_DELAY_NOT_SUPPORTED_WITH_REG_TYPE	This error indicates that the close delay is not supported with this registration type
CTIERR_CONFERENCE_ALREADY_EXISTED	This error indicates that conference already exists
CTIERR_CONFERENCE_NOT_EXISTED	This error indicates that conference does not exist
CTIERR_CONNECTION_ON_INVALID_PORT	This error indicates application is trying to connect to invalid port

Error code	Description
CTIERR_CONSULT_CALL_FAILURE	This error indicates consult call failure
CTIERR_CONSULTCALL_ALREADY_OUTSTANDING	This error indicates that consult call already outstanding
CTIERR_CREATE_PERSISTENT_CALL_FAILED	This error indicates that there is an issue with creating a persistent call.
CTIERR_CRYPTO_CAPABILITY_MISMATCH	This error indicates that device registration failed as device crypto algorithms does not match with current device registration
CTIERR_CTIHANDLER_PROCESS_CREATION_FAILED	This error indicates that CTIHandler process creation failed
CTIERR_DB_INITIALIZATION_ERROR	This error indicates DB initialization error
CTIERR_DEVICE_ALREADY_OPENED	This error indicates that device is already opened
CTIERR_DEVICE_ALREADY_REGISTERED_NONEXTEND	Device registration failed as device is already registered in Non-Extend mode.
CTIERR_DEVICE_NOT_OPENED_YET	This error indicates that device is not yet opened
CTIERR_DEVICE_OWNER_ALIVE_TIMER_STARTED	This error indicates that there is a device registration failure
CTIERR_DEVICE_REGISTRATION_FAILED_ NOT_SUPPORTED_MEDIATYPE	This error indicates an invalid media type, CTIPort need to be registered with Dynamic media port registation if it has an intercom line
CTIERR_DEVICE_RESTRICTED	This error indicates that the device is restricted
CTIERR_DEVICE_SHUTTING_DOWN	This error indicates that device is shutting down
CTIERR_DIRECTORY_LOGIN_TIMEOUT	This error indicates that there is a directory login time out
CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_ CALL_ACTIVE	This error indicates that the request to disconnect the persistent call failed because there is an active customer call. Only when there are no active calls present, can the persistent call be disconnected.
CTIERR_DUPLICATE_CALL_REFERENCE	This error indicates duplicate call reference
CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER	Duplicated Remote Destination Number with an existing Remote Destination Number.
CTIERR_DYNREG_IPADDRMODE_MISMATCH	This indicates registration failure when Cisco Media/Route Tterminal is already registered with different Addressing mode
CTIERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE	Enduser is not associated with the device.
CTIERR_FAC_CMC_REASON_CMC_INVALID	Client Matter Code (CMC) entered is invalid
CTIERR_FAC_CMC_REASON_CMC_NEEDED	CMC is required to offer the call
CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	Forced Authorization Code (FAC) and CMC are required to offer call

Error code	Description
CTIERR_FAC_CMC_REASON_FAC_INVALID	FAC entered is invalid
CTIERR_FAC_CMC_REASON_FAC_NEEDED	FAC is required to offer the call
CTIERR_FEATURE_ALREADY_REGISTERED	This error indicates feature already registered
CTIERR_FEATURE_DATA_REJECT	This error indicates feature data reject
CTIERR_FEATURE_SELECT_FAILED	This error indicates that feature select failed
CTIERR_ILLEGAL_DEVICE_TYPE	This error indicates that the device type is illegal
CTIERR_INCOMPATIBLE_AUTOINSTALL_PROTOCOL_VERSION	This error indicates that auto install protocol version is incompatible
CTIERR_INCORRECT_MEDIA_CAPABILITY	This error indicates that device registration failed due to incorrect media capability
CTIERR_INFORMATION_NOT_AVAILABLE	This error indicates that information is not available
CTIERR_INTERCOM_SPEEDDIAL_ALREADY_CONFIGURED	This error indicates that intercom target value is already configured, application is trying to make call with Intercom target DN
CTIERR_INTERCOM_SPEEDDIAL_ALREADY_SET	This error indicates that intercom request failed as intercom target value is already set, application is trying to set again
CTIERR_INTERCOM_SPEEDDIAL_DESTN_INVALID	This error indicates that intercomm request failed as intercom target value in not in the intercom group
CTIERR_INTERCOM_TALKBACK_ALREADY_PENDING	This error indicates that intercom talk back request is already pending
CTIERR_INTERCOM_TALKBACK_FAILURE	This error indicates that talkback request failed for some reason
CTIERR_INTERNAL_FAILURE	This error indicates there is a CTI internal failure
CTIERR_INVALID_CALLID	This error indicates the call ID is invalid
CTIERR_INVALID_DEVICE_NAME	This error indicates that the device name is not valid
CTIERR_INVALID_DTMFDIGITS	Play DTMF request failed because it is an invalid DTMF digit
CTIERR_INVALID_FILTER_SIZE	This error indicates that filter size is invalid
CTIERR_INVALID_MEDIA_DEVICE	This error indicates that the media device is not valid
CTIERR_INVALID_MEDIA_PARAMETER	This error indicates media parameter is invalid
CTIERR_INVALID_MEDIA_PROCESS	This error indicates that there is an invalid media process
CTIERR_INVALID_MEDIA_RESOURCE_ID	This error indicates media resource ID is not valid
CTIERR_INVALID_MESSAGE_HEADER_INFO	This error indicates that the header info is not valid

Error code	Description
CTIERR_INVALID_MESSAGE_LENGTH	This error indicates that message length is invalid
CTIERR_INVALID_MONITOR_DESTN	This error indicates monitoring request failed due to invalid monitoring destination
CTIERR_INVALID_MONITOR_DN_TYPE	This error indicates an invalid monitor DN type
CTIERR_INVALID_MONITORMODE	This error indicates monitor request failed due to an invalid monitor mode
CTIERR_INVALID_PARAMETER	This error indicates that the parameter is not valid
CTIERR_INVALID_PARK_DN	This error indicates that the DN is an invalid park DN
CTIERR_INVALID_PARK_REGISTRATION_HANDLE	This error indicates that the handle is an invalid park registration handle
CTIERR_PERSISTENT_CALL_EXISTS	This error indicates that a persistent call already exists.
CTIERR_INVALID_REMOTE_DESTINATION_NAME	This error indicates an Invalid Remote Destination Name.
CTIERR_INVALID_REMOTE_DESTINATION_NUMBER	This error indicates an Invalid Remote Destination Number.
CTIERR_INVALID_RESOURCE_TYPE	This error indicates an invalid resource type
CTIERR_IPADDRMODE_MISMATCH	This indicates the registration failure due to IP Addressing Mode mismatch.
CTIERR_LINE_OUT_OF_SERVICE	This error indicates that line is out of service.
CTIERR_LINE_RESTRICTED	This er ror indicates that the line is restricted
CTIERR_MAXCALL_LIMIT_REACHED	This error indicates that maximum call limit has reached
CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC	This error indicates that device registration failed as device is registered with Dynamic media termination
CTIERR_MEDIA_ALREADY_TERMINATED_EXTEND	Device registration failed as device is already registered in Extend mode.
CTIERR_MEDIA_ALREADY_TERMINATED_NONE	This error indicates that device registration failed as device is already registered with media termination none
CTIERR_MEDIA_ALREADY_TERMINATED_STATIC	This error indicates that device registration failed as device is registered with Static media termination
CTIERR_MEDIA_CAPABILITY_MISMATCH	This error indicates that device registration failed as media capability of device does not match with current device registration
CTIERR_MEDIA_RESOURCE_NAME_SIZE_EXCEEDED	This error indicates that media resource name size has exceeded limit
CTIERR_MEDIAREGISTRATIONTYPE_DO_NOT_MATCH	This error indicates that media registration types do not match

Error code	Description
CTIERR_MESSAGE_TOO_BIG	This error indicates that message is too big
CTIERR_MORE_ACTIVE_CALLS_THAN_RESERVED	This error indicates that there are more active calls than reserved
CTIERR_NO_EXISTING_CALLS	This error indicates there are no existing calls
CTIERR_NO_EXISTING_CONFERENCE	This error indicates that there is no existing conference
CTIERR_NO_RECORDING_SESSION	This error indicates recording request failed as there is no recording session
CTIERR_NO_RESPONSE_FROM_MP	This error indicates no response from media resource
CTIERR_NOT_PRESERVED_CALL	This error indicates that the call is not preserved
CTIERR_OPERATION_FAILED_QUIETCLEAR	This error indicates that feature unavailable for this call due to temporary failure
CTIERR_OPERATION_NOT_ALLOWED	This error indicates that this operation is not allowed
CTIERR_OPERATION_NOT_ALLOWED_ON_ PERSISTENT_CALL	This indicates that the specified operation is not allowed on a persistent call.
CTIERR_OUT_OF_BANDWIDTH	This error indicates out of bandwidth error
CTIERR_OWNER_NOT_ALIVE	This error indicates a failure during registering the device
CTIERR_PENDING_ACCEPT_OR_ANSWER_REQUEST	This error indicates that there is a pending accept or answer request
CTIERR_PENDING_START_MONITORING_REQUEST	This error indicates there is a pending start monitoring request
CTIERR_PENDING_START_RECORDING_REQUEST	This error indicates there is a pending start recording request
CTIERR_PENDING_STOP_RECORDING_REQUEST	This error indicates there is a pending stop recording request
CTIERR_PERSISTENT_CALL_BEING_SETUP	This error indicates that the request failed because a persistent call is already being set up.
CTIERR_PRIMARY_CALL_INVALID	This error indicates that primary call in monitoring request in invalid or gone idle
CTIERR_PRIMARY_CALL_STATE_INVALID	This error indicates that primary call in monitoring request is in invalid state
CTIERR_RECORDING_ALREADY_INPROGRESS	This error indicates recording request failed that recording is already in progress
CTIERR_RECORDING_CONFIG_NOT_MATCHING	This error indicates recording configuration does not match
CTIERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING	Stop recording failed because the recording invocation type did not match.
CTIERR_RECORDING_SESSION_INACTIVE	This error indicates recording request failed because recording session is inactive

Error code	Description
CTIERR_REDIRECT_UNAUTHORIZED_COMMAND_USAGE	This error indicates a redirect unauthorized command usage
CTIERR_REGISTER_FEATURE_ACTIVATION_FAILED	This error indicates that register feature activation failed
CTIERR_REGISTER_FEATURE_APP_ALREADY_REGISTERED	Register feature application was already registered
CTIERR_REGISTER_FEATURE_PROVIDER_NOT_REGISTERED	Register feature provider was not registered.
CTIERR_REMOTE_DEVICE_REQUEST_FAILED_ ACTIVE_RD_NOT_SET	The active remote destination is not set.
CTIERR_REMOTEDESTINATION_LIMIT_EXCEEDED	The number of Remote Destinations has exceeded the max number limit.
CTIERR_RESOURCE_NOT_AVAILABLE	This error indicates that resource is not available to fulfill the request
CTIERR_START_MONITORING_FAILED	This error indicates that start monitoring request failed
CTIERR_START_RECORDING_FAILED	This error indicates that start recording request failed
CTIERR_STATION_SHUT_DOWN	This error indicates that there is a station shutdown
CTIERR_SYSTEM_ERROR	This error indicates CTI system error
CTIERR_UDP_PASS_THROUGH_NOT_SUPPORTED	This error indicates UDP data passthrough not supported
CTIERR_UNKNOWN_EXCEPTION	This error indicates an unknown exception occured
CTIERR_UNSUPPORTED_CALL_PARK_TYPE	This error indicates that call park type is not supported
CTIERR_UNSUPPORTED_CFWD_TYPE	This error indicates that the call forward type is unsupported
CTIERR_USER_NOT_AUTH_FOR_SECURITY	This error indicates user is not authorized for secure connection
DARES_INVALID_REQ_TYPE	This error indicates that there is an internal call processing error: DaRes invalid request type
DATA_SIZE_LIMIT_EXCEEDED	This error indicates that XML data object size is bigger than allowed.
DB_ERROR	This error indicates that the device query contained an illegal device type
DB_ILLEGAL_DEVICE_TYPE	This error indicates illegal device type in DB
DB_NO_MORE_DEVICES	This error is no longer used.
DESTINATION_BUSY	This error indicates that destination is busy
DESTINATION_UNKNOWN	This error indicates that destination is not found
DEVICE_ALREADY_REGISTERED	This error indicates that device registration attempt failed, because the device is already registered

Error code	Description
DEVICE_NOT_OPEN	This error indicates that an attempt to open a line failed, as the device is not opened or the device is not registered.
DEVICE_OUT_OF_SERVICE	This error indicates that device is out of service.
DIGIT_GENERATION_ALREADY_IN_PROGRESS	This error indicates that digit generation is already in progress.
DIGIT_GENERATION_CALLSTATE_CHANGED	This error indicates that call state is invalid to continue.
DIGIT_GENERATION_WRONG_CALL_HANDLE	This error indicates that call handle is invalid and call may be gone.
DIGIT_GENERATION_WRONG_CALL_STATE	This error indicates that call state is not valid to generate digits.
DIRECTORY_LOGIN_FAILED	This error indicates that directory login failed: directory not initialized
DIRECTORY_LOGIN_NOT_ALLOWED	This error indicates that directory login failed
DIRECTORY_TEMPORARY_UNAVAILABLE	This error indicates that directory is temporarily unavailable.
EXISTING_FIRSTPARTY	This error indicates that there is already a device controlling media.
HOLDFAILED	This error indicates that the hold was rejected by line control or call control layers
ILLEGAL_CALLINGPARTY	This error indicates that an attempt was made to originate call using a calling party that is not on the device
ILLEGAL_CALLSTATE	This error indicates line is not in a legal state to invoke the request
ILLEGAL_HANDLE	This error indicates the handle is not valid
ILLEGAL_MESSAGE_FORMAT	This error indicates that there is a QBE protocol error
INCOMPATIBLE_PROTOCOL_VERSION	This error indicates that JTAPI and CTI versions are not compatible : CTI Error Protocol version not supported
INVALID_LINE_HANDLE	This error indicates that attempt to perform a line operation on an invalid line handle.
INVALID_RING_OPTION	This error indicates that the ring option is invalid
LINE_GREATER_THAN_MAX_LINE	This error indicates that line is greater than the maximum available lines on this device
LINE_INFO_DOES_NOT_EXIST	This error indicates that line information does not exist in the database.
LINE_NOT_PRIMARY	This error indicates that internal error returned from call control.
LINECONTROL_FAILURE	This error indicates line control refuses to allow a new call to be initiated because of its current state.

Error code	Description
MAX_NUMBER_OF_CTI_CONNECTIONS_REACHED	The maximum number of CTI connections was reached.
MSGWAITING_DESTN_INVALID	This error indicates that attempt to set message waiting lamp for an invalid DN; Message Waiting Destination not found.
NO_ACTIVE_DEVICE_FOR_THIRDPARTY	This error indicates there is no active device for thirdparty
NO_CONFERENCE_BRIDGE	This error indicates that no conference bridge available
NOT_INITIALIZED	This error indicates that attempt is made to open a provider before CTI initialization completes
PROTOCOL_TIMEOUT	Internal error returned from call control
PROVIDER_ALREADY_OPEN	This error indicates that an attempt is made to reopen provider
PROVIDER_CLOSED	This error indicates an attempt to close provider while it is already closed
PROVIDER_NOT_OPEN	This error indicates that device list incomplete or device list query timeout or query aborted
REDIRECT_CALL_CALL_TABLE_FULL	This error indicates that internal error is returned from call control
REDIRECT_CALL_DESTINATION_BUSY	This error indicates that the redirect destination is busy
REDIRECT_CALL_DESTINATION_OUT_OF_ORDER	This error indicates that redirect destination is out of order
REDIRECT_CALL_DIGIT_ANALYSIS_TIMEOUT	This error indicates a digit analyss time out, this is an internal error returned from call control
REDIRECT_CALL_DOES_NOT_EXIST	This error indicates that an attempt is made to redirect a call that does not exist or is not longer active
REDIRECT_CALL_INCOMPATIBLE_STATE	This error indicates that internal error is returned from call control
REDIRECT_CALL_MEDIA_CONNECTION_FAILED	This error indicates media connection failure, this is an internal error returned from call control
REDIRECT_CALL_NORMAL_CLEARING	This error indicates that redirect failed because of normal call clearing
REDIRECT_CALL_ORIGINATOR_ABANDONED	This error indicates that far end hung up on the call being redirected
REDIRECT_CALL_PARTY_TABLE_FULL	This error indicates that internal error is returned from call control
REDIRECT_CALL_PENDING_REDIRECT_TRANSACTION	This error indicates that internal error is returned from call control
REDIRECT_CALL_PROTOCOL_ERROR	This error indicates a protocol error, this is an internal error returned from call control
REDIRECT_CALL_UNKNOWN_DESTINATION	This error indicates that an attempt is made to redirect to an unknown destination

Troubleshooting Cisco Unified JTAPI

Error code	Description
REDIRECT_CALL_UNKNOWN_ERROR	This error indicates that internal error is returned from call control
REDIRECT_CALL_UNKNOWN_PARTY	This error indicates an unknown party is detected, this is an internal error returned from call control
REDIRECT_CALL_UNRECOGNIZED_MANAGER	This error indicates that internal error is returned from call control
REDIRECT_CALLINFO_ERR	This error indicates that internal error is returned from call control
REDIRECT_ERR	This error indicates that internal error is returned from call control
RETRIEVEFAILED	This error indicates that retrieval of call was rejected by line control or call control
RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	This error indicates that error occurred in retrieving held call; because there is already another active call on the line
SSAPI_NOT_REGISTERED	This error indicates that the redirect command was issued when the internal supporting interface was not initialized; either CTI has not yet finished its initialization or an internal error occurred
TIMEOUT	This error indicates that the request has timed out.
TRANSFER_INACTIVE	This error indicates that attempt to complete transfer, while consult tranfer is not there
TRANSFERFAILED	This error indicates that the transfer failed probably because one of the call legs was hung up or disconnected from the far end
TRANSFERFAILED_CALLCONTROL_TIMEOUT	This error indicates that expected response from call control not received during a transfer
TRANSFERFAILED_DESTINATION_BUSY	This error indicates that an attempt is made to transfer call to a busy destination
TRANSFERFAILED_DESTINATION_UNALLOCATED	This error indicates an attempt is made to to transfer call to a directory number that is not registered
TRANSFERFAILED_OUTSTANDING_TRANSFER	This error indicates that existing transfer is still in progress
UNDEFINED_LINE	This error indicates that the line that was specified, is not found on the device
UNKNOWN_GLOBAL_CALL_HANDLE	This error indicates that the global call handle is unknown
UNRECOGNIZABLE_PDU	This error indicates that there is a QBE protocol error
UNSPECIFIED	This error indicates that an unspecified error has occurred.

CiscoEventIDs

This section includes the following events:

Provider Events

Event name	Event number
CiscoProvFeatureUnRegisteredEv	0x40000008
CiscoRestrictedEv	0x40000009
CiscoAddrRestrictedEv	0x40000010
CiscoTermRestrictedEv	0x40000011
CiscoAddrActivatedEv	0x40000012
CiscoTermActivatedEv	0x40000013
CiscoAddrActivatedOnTerminalEv	0x40000014
CiscoAddrRestrictedOnTerminalEv	0x40000015
CiscoProviderCapabilityChangedEv	0x40000016
CiscoProvTerminalCapabilityChangedEv	0x40000017
CiscoProvTerminalRegisteredEv	0x40000018
CiscoProvTerminalUnRegisteredEv	0x40000019
CiscoProvTerminalRemoteDestinationChangedEv	0x40000020
CiscoProvTerminalIPAddressChangedEv	0x40000021
CiscoProvTerminalMultiMediaCapabilityChangedEv	0x40000022

Terminal Events

Event name	Event number
CiscoTermCreatedEv	0x40001001
CiscoTermDataEv	0x40001002
CiscoTermInServiceEv	0x40001003
CiscoTermOutOfServiceEv	0x40001004
CiscoTermRemovedEv	0x40001005
CiscoTermDeviceActiveStatusEv	0x40001006
CiscoTermDeviceAlertingStatusEv	0x40001007
CiscoTermDeviceHoldStatusEv	0x40001008
CiscoTermDeviceIdleStatusEv	0x40001009
CiscoTermButtonPressedEv	0x40001010
CiscoTermRegistraionFailedEv	0x40001011
CiscoTermDNDStatusChangedEv	0x40001014
CiscoTermDeviceStateWhisperEv	0x40001015
CiscoTermDNDOptionChangedEv	0x40001016
CiscoMultiMediaStreamsInfoEv	0x40001017

Address Events

Event name	Event number
CiscoAddrCreatedEv	0x40002001
CiscoAddrInServiceEv	0x40002002
CiscoAddrOutOfServiceEv	0x40002003
CiscoAddrRemovedEv	0x40002004
CiscoOutOfServiceEv	0x40002005
CiscoAddrAddedToTerminalEv	0x40002006
CiscoAddrRemovedFromTerminalEv	0x40002007
CiscoAddrAutoAcceptStatusChangedEv	0x40002008

Event name	Event number
CiscoAddrIntercomInfoChangedEv	0x40002009
CiscoAddrIntercomInfoRestorationFailedEv	0x40002010
CiscoAddrRecordingConfigChangedEv	0x40002011
CiscoAddrParkStatusEv	0x40002012
CiscoAddrVoiceMailPilotChangedEv	0x40002013
CiscoAddrPickupGroupChangedEv	0x40002014
CiscoAddrMonitoringTerminatedEv	0x4000200A

Call Events

Event name	Event number
CiscoProvCallParkEv	0x40003001
CiscoConferenceEndEv	0x40003002
CiscoConferenceStartEv	0x40003003
CiscoConsultCallActiveEv	0x40003004
CiscoTransferEndEv	0x40003005
CiscoTransferStartEv	0x40003006
CiscoToneChangedEv	0x40003007
CiscoCallChangedEv	0x40003008
CiscoConferenceChainAddedEv	0x40003009
CiscoConferenceChainRemovedEv	0x40003010
CiscoCallSecurityStatusChangedEv	0x40003011
CiscoCallFeatureCancelledEv	0x40003012
CiscoProvPickupCallAlertEv	0x40003013
CiscoProvPickupNotificationRegistrationClosedEv	0x40003014
CiscoCallInfoChangedEv	0x40003015
CiscoProvAuthenticationInfoEv	0x40003016

RTP Events

Event name	Event number
CiscoRTPInputStartedEv	0x40004001
CiscoRTPInputStoppedEv	0x40004002
CiscoRTPOutputStartedEv	0x40004003
CiscoRTPOutputStoppedEv	0x40004004
CiscoMediaOpenLogicalChannelEv	0x40004005
CiscoRTPInputKeyEv	0x40004006
CiscoRTPOutputKeyEv	0x40004007
CiscoMediaOpenIPPortEv	0x40004008

TermConn Events

Event name	Event number
CiscoTermConnPrivacyChangedEv	0x40005001
CiscoCallCtlTermConnHeldReversionEv	0x40005002
CiscoTermConnSelectChangedEv	0x40005003
CiscoTermConnRecordingStartEv	0x40005004
CiscoTermConnRecordingEndEv	0x40005005
CiscoTermConnRecordingFailedEv	0x4000500E
CiscoTermConnMonitoringStartEv	0x40005006
CiscoTermConnMonitoringEndEv	0x40005007
CiscoTermConnRecordingTargetInfoEv	0x40005008
CiscoTermConnMonitorInitiatorInfoEv	0x40005009
CiscoTermConnMonitorTargetInfoEv	0x4000500A
CiscoTermConnMonitorUpdatedEv	0x4000500B
CiscoMediaStreamStartedEv	0x4000500C
CiscoMediaStreamEndedEv	0x4000500D

Conn Events

Event name	Event number
CiscoConnectionUniqueIDChangedEv	0x40006001
CiscoHuntConnCreatedEv	0x40006002

Reason Codes

The following codes are defined in CiscoFeatureReason interface.

Reason code name	Reason code
REASON_TRANSFER	2
REASON_FORWARDNOANWSER	3
REASON_FORWARDBUSY	4
REASON_FORWARDALL	5
REASON_REDIRECT	6
REASON_BLINDTRANSFER	7
REASON_CONFERENCE	9
REASON_PARK	10
REASON_CALLPICKUP	11
REASON_NORMAL	12
REASON_PARKREMINDER	15
REASON_UNPARK	16
REASON_BARGE	20
REASON_IMMDIVERT	21
REASON_FAC_CMC	22
REASON_QSIG_PR	23
REASON_REFER	24
REASON_REPLACE	25
REASON_CCM_REDIRECTION	26
REASON_DPARK_CALLPARK	27

Reason code name	Reason code
REASON_DPARK_REVERSION	28
REASON_DPARK_UNPARK	29
REASON_SILENTMONITORING	31
REASON_MOBILITY	33
REASON_MOBILITY_IVR	34
REASON_MOBILITY_CELLPICKUP	35
REASON_MOBILITY_HANDIN	36
REASON_MOBILITY_HANDOUT	37
REASON_MOBILITY_FOLLOWME	38
REASON_CLICK_TO_CONFERENCE	39
REASON_FORWARD_NO_RETRIEVE	40
REASON_EXTERNALCALLCONTROL	42
REASON_SAF_CCD_PSTN_FAILOVER	43
REASON_MEDIA_STREAMING	44

Cause Codes

Cause code name	Cause code
CAUSE_NOERROR	0X00 (0)
CAUSE_UNALLOCATEDNUMBER	0X01 (1)
CAUSE_NOROUTETOTRANSITNET	0X02 (2)
CAUSE_NOROUTETODDESTINATION	0X03 (3)
CAUSE_CHANUNACCEPTABLE	0X06 (6)
CAUSE_CALLBEINGDELIVERED	0X07 (7)
CAUSE_CTIPREEMPTNOREUSE	0X08 (8)
CAUSE_CTIPREEMPTFORREUSE	0X09 (9)
CAUSE_NORMALCALLCLEARING	0X10 (16)
CAUSE_USERBUSY	0X11 (17)

Cause code name	Cause code
CAUSE_NOUSERRESPONDING	0X12 (18)
CAUSE_NOANSWERFROMUSER	0X13 (19)
CAUSE_SUBSCRIBERABSENT	0X14 (20)
CAUSE_CALLREJECTED	0X15 (21)
CAUSE_NUMBERCHANGED	0X16 (22)
CAUSE_EXCHANGEROUTINGERROR	0X19 (25)
CAUSE_NONSELECTEDUSERCLEARING	0X1A (26)
CAUSE_DDESTINATIONOUTOFORDER	0X1B (27)
CAUSE_INVALIDNUMBERFORMAT	0X1C (28)
CAUSE_FACILITYREJECTED	0X1D (29)
CAUSE_RESPONSETOSTATUSENQUIRY	0X1E (30)
CAUSE_NORMALUNSPECIFIED	0X1F (31)
CAUSE_NOCIRCAVAIL	0X22 (34)
CAUSE_NETOUTOFORDER	0X26 (38)
CAUSE_TEMPORARYFAILURE	0X29 (41)
CAUSE_SWITCHINGEQUIPMENTCONGESTION	0X2A (42)
CAUSE_ACCESSINFORMATIONDISCARDED	0X2B (43)
CAUSE_REQCIRCNAVIL	0X2C (44)
CAUSE_CTIPRECEDENCECALLBLOCKED	0X2E (46)
CAUSE_RESOURCESNAVAIL	0X2F (47)
CAUSE_QUALOFSERVNAVAIL	0X31 (49)
CAUSE_REQFACILITYNOTSUBSCRIBED	0X32 (50)
CAUSE_SERVOPERATIONVIOLATED	0X35 (53)
CAUSE_INCOMINGCALLBARRED	0X36 (54)
CAUSE_BCNAUTHORIZED	0X39 (57)
CAUSE_BCBPRESENTLYAVAIL	0X3A (58)
CAUSE_SERVNOTAVAILUNSPECIFIED	0X3F (63)
CAUSE_BEARERCAPNIMPL	0X41 (65)

Cause code name	Cause code
CAUSE_CHANTYPENIMPL	0X42 (66)
CAUSE_REQFACILITYNIMPL	0X45 (69)
CAUSE_ONLYRDIVEARERCAPAVAIL	0X46 (70)
CAUSE_SERVOROPTNAVAILORIMPL	0X4F (79)
CAUSE_INVALIDCALLREFVALUE	0X51 (81)
CAUSE_IDENTIFIEDCHANDOESNOTEXIST	0X52 (82)
CAUSE_SUSPCALLBUTNOTTHISONE	0X53 (83)
CAUSE_CALLIDINUSE	0X54 (84)
CAUSE_NOCALLSUSPENDED	0X55 (85)
CAUSE_REQCALLIDHASBEENCLEARED	0X56 (86)
CAUSE_INCOMPATABLEDDESTINATION	0X58 (88)
CAUSE_DESTNUMMISSANDDCNOTSUB	0X5A (90)
CAUSE_INVALIDTRANSITNETSEL	0X5B (91)
CAUSE_INVALIDMESSAGEUNSPECIFIED	0X5F (95)
CAUSE_MANDATORYIEMISSING	0X60 (96)
CAUSE_MSGTYPENIMPL	0X61 (97)
CAUSE_MSGTYPENCOMPATWCS	0X62 (98)
CAUSE_IENIMPL	0X63 (99)
CAUSE_INVALIDIECONTENTS	0X64 (100)
CAUSE_MSGNCOMPATABLEWCS	0X65 (101)
CAUSE_RECOVERYONTIMEREXPIRY	0X66 (102)
CAUSE_PROTOCOLERRORUNSPECIFIED	0X6F (111)
CAUSE_CTIPRECEDENCELEVELEXCEEDED	0X7A (122)
CAUSE_CTIDEVICENOTPREEMPTABLE	0X7B (123)
CAUSE_OUTOFBANDWIDTH	0X7D (125)
CAUSE_INTERWORKINGUNSPECIFIED	0X7F (127)
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH	0X81 (129)
CAUSE_REDIRECTED	0XC9 (200)

Cause code name	Cause code
CAUSE_INTERNALCAUSE	0X1F4 (500)
CAUSE_OUTBOUND_TRANSFER	0X1F5 (501)
CAUSE_OUTBOUND_CONFERENCE	0X1F6 (502)
CAUSE_INBOUND_TRANSFER	0X1F7 (503)
CAUSE_INBOUND_CONFERENCE	0X1F8 (504)
CAUSE_INBOUND_BLINDTRANSFER	0X1F9 (505)
CAUSE_CTIMANAGER_FAILURE	0x1FB (507)
CAUSE_CALLMANAGER_FAILURE	0x1FC (508)
CAUSE_BARGE	0x1FD(509)
CAUSE_FAC_CMC	0x1FE (510)
CAUSE_QSIG_PR	0x1FF (511)
CAUSE_DPARK	0x200 (512)
CAUSE_DPARK_UNPARK	0x201 (513)
CAUSE_DPARK_REMINDER	0x202 (514)
CAUSE_QUIET_CLEAR	0x203 (515)
CAUSE_CTICONFERENCEFULL	0X40000 + CAUSE_NOERROR
CAUSE_CALLSPLIT	0X60000 + CAUSE_NOERROR
CAUSE_CTIDROPCONFEREE	0X70000 + CAUSE_NOERROR
CAUSE_CTICCMSIP400BADREQUEST	0X1000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP401UNAUTHORIZED	0X2000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP402PAYMENTREQUIRED	0X3000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP403FORBIDDEN	0X4000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP404NOTFOUND	0X5000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP405METHODNOTALLOWED	0X6000000 + CAUSE_SERVNOTAVAILUNSPECIFIED
CAUSE_CTICCMSIP406NOTACCEPTABLE	0X7000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED	0X8000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP408REQUESTTIMEOUT	0X9000000 + CAUSE_RECOVERYONTIMEREXPIRY
CAUSE_CTICCMSIP410GONE	0XB000000 + CAUSE_NUMBERCHANGED

Cause code name	Cause code
CAUSE_CTICCMSIP411LENGTHREQUIRED	0XC000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG	0XE000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP414REQUESTURITOOLONG	0XF000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE	0X10000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME	0X11000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP420BADEXTENSION	0X15000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP421EXTENSTIONREQUIRED	0X16000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP423INTERVALTOOBRIEF	0X18000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE	0X40000000 + CAUSE_NOUSERRESPONDING
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST	0X41000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP482LOOPDETECTED	0X42000000 + CAUSE_EXCHANGEROUTINGERROR
CAUSE_CTICCMSIP483TOOMANYHOOPS	0X43000000 + CAUSE_EXCHANGEROUTINGERROR
CAUSE_CTICCMSIP484ADDRESSINCOMPLETE	0X44000000 + CAUSE_INVALIDNUMBERFORMAT
CAUSE_CTICCMSIP485AMBIGUOUS	0X45000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP486BUSYHERE	0X46000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP487REQUESTTERMINATED	0X47000000 + CAUSE_NORMALUNSPECIFIED
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE	0X48000000 + CAUSE_NORMALUNSPECIFIED
CAUSE_CTICCMSIP491REQUESTPENDING	0X4B000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP493UNDECIPHERABLE	0X4D000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP500SERVERINTERNALERROR	0X54000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP501NOTIMPLEMENTED	0X55000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP502BADGATEWAY	0X56000000 + CAUSE_NETOUTOFORDER
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE	0X57000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP504SERVERTIMEOUT	0X58000000 + CAUSE_RECOVERYONTIMEREXPIRY
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED	0X59000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP513MESSAGETOOLARGE	0X5A000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP600BUSYEVERYWHERE	0XA1000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP603DECLINE	0XA2000000 + CAUSE_CALLREJECTED

Cause code name	Cause code
CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE	0XA3000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP606NOTACCEPTABLE	0XA4000000 + CAUSE_NORMALUNSPECIFIED
CAUSE_CTICCMSIP200CALLCOMPLETEDELSEWHERE	0xA5000000 + NORMALUNSPECIFIED
CAUSE_CTICCMSIP503SERVICENOTAVAILABLE	0xA7000000 + SERVNOTAVAILUNSPECIFIED

Additional Troubleshooting Information

Viewing JTAPI Debug Output

To view JTAPI debug output, use the JTPREFS application to change the trace settings. The JTPREFS application allows you to enable or disable various kinds of tracing.

JTPREFS is installed in the **%SystemRoot%\java\lib** directory along with the JTAPI classes. Cisco JTAPI Preferences is installed by default in **Program Files\JTAPITools**.

To open the Cisco JTAPI Preferences utility, choose **Start > Programs > Cisco JTAPI > JTAPI** Preferences.

The following trace levels are defined:

- WARNING - warning events
- INFORMATIONAL - status events
- DEBUG - debugging events

If DEBUG is enabled, JTPREFS allows you to enable or disable various debugging levels.

The following debugging levels are defined:

- TAPI_DEBUGGING - to trace JTAPI methods and events
- TAPI_IMPLDEBUGGING - internal JTAPI implementation trace
- CTI_DEBUGGING - to trace Cisco Unified Communications Manager events that are sent to the JTAPI implementation
- CTIIMPL_DEBUGGING - internal CTICLIENT implementation trace
- PROTOCOL_DEBUGGING - full CTI protocol decoding
- MISC_DEBUGGING - miscellaneous low-level debug trace

Traces can be directed to a specific path and folder rather than to the application directory by default. The same trace folder could be used for successive or more than one simultaneous launch of JTAPI. Different launches of JTAPI would also send the traces to different folders. This allows simultaneous JTAPI instances to maintain independent trace destinations

Note Traces can be directed to a specific path and folder rather than to the application directory by default. The same trace folder could be used for successive or more than one simultaneous launch of JTAPI. Different launches of JTAPI would also send traces to different folders. This allows simultaneous JTAPI instances to maintain independent trace destinations. The application directory in this case is not that of the JTAPI client itself, but of the application that is integrating/using the JTAPI client.

Log Files for JTAPI Client Installer

In order to detect the error which might occur during the installation and uninstallation process, two log files will be generated. These files will be in the same location from which the installer is executed.

- ismpInstall.log – to track events during installation.
- ismpUninstall.log – to track events during uninstallation.

The error messages will contain the information about the wizard beans that were executed as a part of the install procedure and if there were any exceptions.

Troubleshooting Tips for ISMP Installer

SN	Problem Description	Cause	Solution
1	ISMP Uninstall does not remove the target directories installed.	Directory from which uninstaller is invoked.	The uninstaller needs to be invoked from at least one level above the install directory.
2	Proper language details are not displayed during installation	Locale Files not proper.	Please report this problem immediately to the support personnel to suggest the change or error in message.
3	Uninstaller/Installer throws error.	The JVM has been either removed or replaced with an incompatible version	The installer comes with a built in JVM which also gets installed if the target machine does not have a JVM. In case you face this error - manual removal of the files needs to be done.
4	Installer goes through fine, but the files have not been copied.	Permissions	Ensure that proper write permissions are there for the destination folder. This problem can occur on UNIX platforms.
5	Installer/Uninstaller throws exception or crashes during the installation process.	version name problem / folder name problem.	Refer to the log files generated to get an idea of which step caused the error.

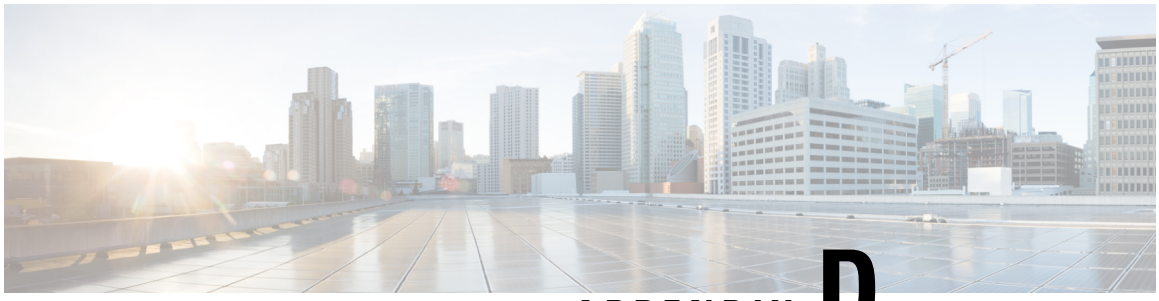
SN	Problem Description	Cause	Solution
6	Upgrade does not show “upgrade” message during installation of an upgrade version.	.jtapiver.ini missing.	This file is where the current jtapi install details are located. If this is accidently removed then, upgrade/reinstall will have display issues. In the case of an upgrade/reinstall or downgrade failure, the user will have to manually remove the files from the .jtapi/bin and .jtapi/lib folders and then try the installer in order to ensure proper installation during the next time.

Unable to Create Provider Directory Login Timeout

This error occurs when there is no authentication response from CTI for the ProviderOpenRequest. It could fail because of:

- LDAP connectivity problems
- Database delays
- The CTIManager being busy for some other reason and therefore unable to honor the request

The solution is that the application must try again. If the ProviderOpenRequest fails on repeated attempts, modify the ProviderOpenRequest.



APPENDIX D

Cisco Unified JTAPI Operations by Release

This section lists supported, unsupported, changed, and "under consideration or review" features for Cisco Unified JTAPI by Cisco Unified Communications Manager release. The details can be found in the Cisco Unified Communications Manager JTAPI Developers Guide at http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

JTAPI Operations-by-Release

Table legend:

s: supported, N: not supported, M: Modified, UCR: Under Consideration or Review.

Table 359: JTAPI Features by Cisco Unified Communications Manager Release

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
CTI Manager and Support for fault tolerance	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for Cisco CallManager Extension Mobility	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Blind Transfer (using Redirect)	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support Forward	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Reset the Original Called Party with Redirect	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
CiscoAddr InServiceEv or CiscoAddr OutOFServiceEv	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

Cisco Unified JTAPI Operations by Release

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
Localization / Internationalization	N	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
User Deletion from Directory	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Park and Unpark	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Monitoring Call Park Numbers	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Call Reason Enhancements	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Device Data Passthrough	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
CiscoJTAPI Auto Install	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Multiple Calls per DN	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Shared Line Support	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Transfer	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Direct Transfer	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Conference	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Join	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Privacy Release	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Barge and cBarge	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Dynamic Port Registration	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Media Termination at Route Points	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Transfer to VoiceMail	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Modifying Calling Number	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for Presentation Indication	N	N	N	s	s	s	s	s	M	s	s	s	s	s	s	s	s	s	s
QSIG-PR	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
FAC/CMC Support	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Device State Server	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
SuperProvider Functionality	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Windows 2003 Support	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Directed PARK	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Forward on NoBandWidth and Unregister	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
VoiceMailBox Support	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Privacy on Hold	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Hold Reversion (4.2.1.SR1)	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for MLPP (4.2.2)	s	s	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s
Conference Enhancement-Add Participants to Conf by Non-controller (4.2.2)	N	N	N	N	N	s	s	N	N	s	s	s	s	s	s	s	s	s	s
Conference Chaining (4.2.2)	N	N	N	N	N	s	s	N	N	s	s	s	s	s	s	s	s	s	s
CiscoRTPHandle Interface (4.2.2)	N	N	N	N	N	s	s	N	N	s	s	s	s	s	s	s	s	s	s
CiscoTermRegistration FailedEv- New Error Code (4.2.3)	N	N	N	s	s	M	s	M	s	s	s	s	s	s	s	s	s	s	s
Network events	s	s	s	s	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s
BWC Enhancement	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Hairpin Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Unicode Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SRTP support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s

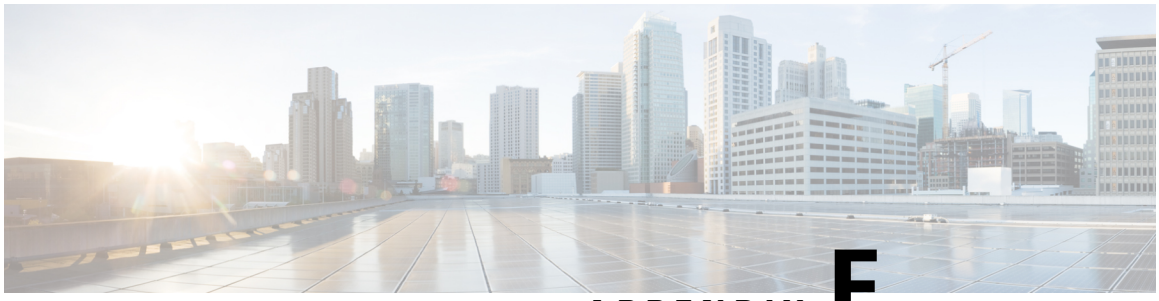
Cisco Unified JTAPI Operations by Release

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
Partition Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Security (TLS) support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Alternate Script Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SIP Features Refer/Replaces	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SIP End Point Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Change Notification of SuperProvider and CallParkDN Monitoring capability	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
3XX	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Call Select Status	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
QoS support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Linux and Solaris Installer	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Intercom Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s
Secure Conferencing Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s
Monitoring & Recording	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s
Arabic and Hebrew Language Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s
Do-Not_Disturb Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s
Join AcrossLine (SCCP)	N	N	N	N	N	N	N	s	s	N	s	s	s	s	s	s	s	s	s
Certificate Download API Enhancement	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Intercom Support for Extension Mobility	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Join Across Line (SIP phone support)	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
Locale Infrastructure Enhancement	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
DND-CallReject (DND-R)	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Call Party Normalization (CPN)	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Click-To-Conference	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
IPv6 Support	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Windows Vista Support	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s
EMLogin UserName API	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
SetJtapiProperties API on CiscoJTAPIPeer	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
DropAnyParty from Conference	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Swap/Cancel - Transfer/Conference Behavior Change	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Direct Transfer Across Lines	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Park Monitoring enhancements	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Assisted DPark	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Enhanced MWI	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Logical Partitioning	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Rollover Support (6921 and 7931)	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Address and Terminal Settings (max calls, voice mail, busy trigger etc.)	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s

Cisco Unified JTAPI Operations by Release

JTAPI features	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1	7.1.3	8.0	8.5	8.6	9.0	10.0
End to End Call Tracing	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Extension Mobility Cross Cluster	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Hunt List Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Call Pickup Invocation	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
External Call Control	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
CallFwdAll Key Press Notification	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Agent Greeting	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Whisper Coaching	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Agent Zip Tone	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Early Offer (Session Manager)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Single Sign On (SSO)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Energywise (Deep Sleep)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
UCR 2008 / FIPS Compliance	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
Windows 7 Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
64-Bit Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
Cisco Extend & Connect (CTI Remote Device)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
Recording Key Enhancement	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
Native Queuing	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
E911 Teleworker	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
Cius Persistency	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
CTI Video Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
Gateway Recording	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s



APPENDIX E

CTI Supported Devices

The following table provides information about CTI supported devices.

You can see the latest list of Cisco CTI Supported Devices at http://developer.cisco.com/web/jtapi/wikidocs

• CTI Supported Devices Table, on page 1655

CTI Supported Devices Table

Table legend:

✓: supported, ✗ : not supported, NA: Not Applicable.

Table 360: CTI Supported Device Matrix

Device/Phone model	SCCP	SIP	Comments
Analog Phone	✓	✗	You can find information on the limitations of this device in Cisco JTAPI Developer Guide for Cisco Unified CallManager 4.1(3) .
Cisco 30 SP+	✓	✗	End of Software Maintenance Release 2001
Cisco 6901	✓	✓	SIP devices require firmware update 9.1(1) available on Cisco.com
Cisco 6911	✓	✓	SIP devices require firmware update 9.1(1) available on Cisco.com
Cisco 6921	✓	✓	PhoneSetDisplay() interface is not supported. SIP devices require firmware update 9.1(1) available on Cisco.com.

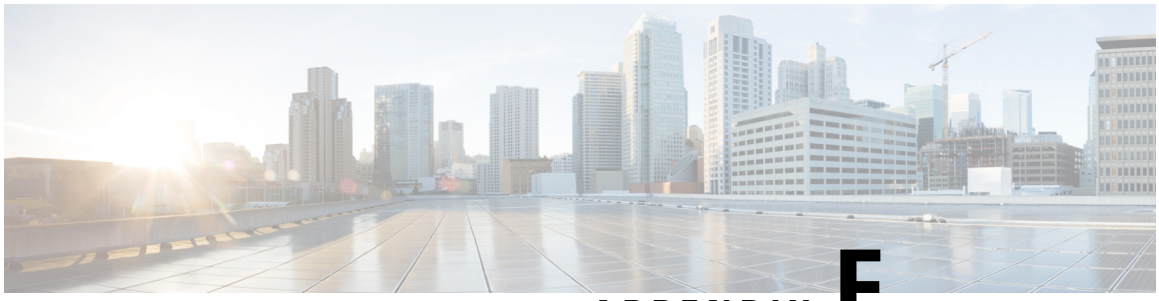
Device/Phone model	SCCP	SIP	Comments
Cisco 6941	✓	✓	PhoneSetDisplay() interface is not supported. SIP devices require firmware update 9.1(1) available on Cisco.com.
Cisco 6945	✓	✓	PhoneSetDisplay() interface is not supported. SIP devices require firmware update 9.1(1) available on Cisco.com.
Cisco 6961	✓	✓	PhoneSetDisplay() interface is not supported. SIP devices require firmware update 9.1(1) available on Cisco.com.
Cisco 7906	✓	✓	
Cisco 7911	✓	✓	
Cisco 7914 Sidecar	✓	✗	End of Software Maintenance Release 2010
Cisco 7915 Sidecar	✓	✓	
Cisco 7916 Sidecar	✓	✓	
Cisco CKEM Sidecar	✗	✓	
Cisco 7921	✓	✗	
Cisco 7925 & 7925-EX	✓	✗	
Cisco 7931	✓	✗	CTI supported only if rollover is disabled. Starting withrelease 7.1 this device is supported when corresponding role is added to user.
Cisco 7936	✓	✗	End of Software Maintenance Release 2011
Cisco 7937	✓	✗	
Cisco 7940	✓	✗	End of Software Maintenance Release 2011
Cisco 7941	✓	✓	
Cisco 7941G-GE	✓	✓	End of Software Maintenance Release 2009

Device/Phone model	SCCP	SIP	Comments
Cisco 7942	✓	✓	
Cisco 7945	✓	✓	
Cisco 7960	✓	✗	End of Software Maintenance Release 2011
Cisco 7961	✓	✓	
Cisco 7961G-GE	✓	✓	End of Software Maintenance Release 2009
Cisco 7962	✓	✓	
Cisco 7965	✓	✓	
Cisco 7970	✓	✓	End of Software Maintenance Release 2009
Cisco 7971	✓	✓	End of Software Maintenance Release 2009
Cisco 7975	✓	✓	
Cisco 7985	✓	✗	End of Software Maintenance Release 2011
Cisco 8811	✗	✓	8811 phones are CTI controlled
Cisco 8941	✓	✗	
Cisco 8945	✓	✗	
Cisco 8961	✗	✓	phoneSetDisplay() interface is not supported
Cisco 9951	✗	✓	phoneSetDisplay() interface is not supported
Cisco 9971	✗	✓	phoneSetDisplay() interface is not supported
Cisco ATA 186	✓	✗	You can find information on the limitations of this device in Cisco JTAPI Developer Guide for Cisco Unified CallManager 4.1(3) .

Device/Phone model	SCCP	SIP	Comments
Cisco Cius	✗	✓	CTI support added in release 8.5(1) phoneSetDisplay() interface is not supported XSI interface is not supported. Silent Monitoring/Recording is not supported
Cisco IP Communicator	✓	✓	CTI support added in release 7.1(2)
Cisco Jabber for Windows (Softphone Mode)	✗	✓	Requires Jabber 9.0
Cisco Jabber for Windows (Extend/Connect Mode)	✗	✗	Supports **CTI**. Requires CUCM 9.1(1a) and Jabber 9.1(2)
Cisco Jabber for Windows (Remote Desktop Control Mode)	—	—	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Jabber for Mac (Softphone Mode)	✗	✓	Requires CUCM 8.6(1)
Cisco Jabber for iPhone & iPad	✗	✓	Support for CTI event monitoring added in CUCM 12.5 su1 for WiFi mode only. Does not support invoking call control/feature requests. See Release Notes for details
Cisco Jabber for Android	✗	✓	Support for CTI event monitoring added in CUCM 12.5 su1 for WiFi mode only. Does not support invoking call control/feature requests. See Release Notes for details
Cisco Unified Personal Communicator - Softphone Mode	✓	✓	CTI support added in release 8.5(1)
Cisco Unified Personal Communicator - Remote Desktop Control Mode	—	—	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.

Device/Phone model	SCCP	SIP	Comments
Cisco Unified Communicator Integration for Microsoft Office Communicator/Lync - Softphone Mode	✗	✓	CTI support added in release 8.5(2)
Cisco Unified Communicator Integration for Microsoft Office Communicator/Lync - Remote Desktop Control Mode	—	—	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Web Communicator for Quad - Softphone Mode	—	—	Not a CTI supported device.
Cisco Web Communicator for Quad - Remote Desktop Control Mode	—	—	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Unified Communications Integration for Webex Connect - Softphone Mode	—	—	Not a CTI supported device.
Cisco Unified Communications Integration for Webex Connect - Remote Desktop Control Mode	—	—	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco VGC Phone	✓	✗	
VG224	—	—	Not a CTI supported device.
VG248	✓	✗	You can find information on the limitations of this device in Cisco JTAPI Developer Guide for Cisco Unified CallManager 4.1(3) .
CTI Port	—	—	CTI supported device that does not use SCCP or SIP.
CTI Remote Device	—	—	CTI supported device that does not use SCCP or SIP.
CTI Route Point	—	—	CTI supported device that does not use SCCP or SIP.

Device/Phone model	SCCP	SIP	Comments
CTI Route Point (Pilot Point)	—	—	CTI supported device that does not use SCCP or SIP.
ISDN BRI Phone	—	—	Not a CTI supported device.
Cisco Spark remote device	—	—	Not supported by CTI



APPENDIX F

Constant Field Values

This appendix lists the static final fields and their values.

com.cisco.*

CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedEv		
public static final int	ID	1073741842

CiscoAddrActivatedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv		
public static final int	ID	1073741844

CiscoAddrAddedToTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv		
public static final int	ID	1073750022

CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv		
public static final int	ID	1073750024

CiscoAddrCreatedEv

com.cisco.jtapi.extensions.CiscoAddrCreatedEv		
public static final int	ID	1073750017

CiscoAddress

com.cisco.jtapi.extensions.CiscoAddress		
public static final int	AUTO_RECORDING	1
public static final int	AUTOACCEPT_OFF	0
public static final int	AUTOACCEPT_ON	1
public static final int	AUTOANSWER_OFF	0
public static final int	AUTOANSWER_UNKNOWN	3
public static final int	AUTOANSWER_WITHHEADSET	1
public static final int	AUTOANSWER_WITHSPEAKERSET	2
public static final int	EXTERNAL	2
public static final int	EXTERNAL_UNKNOWN	3
public static final int	IN_SERVICE	1
public static final int	INTERNAL	1
public static final int	MONITORING_TARGET	5
public static final int	NO_RECORDING	0
public static final int	OUT_OF_SERVICE	0
public static final int	RINGER_DEFAULT	0
public static final int	RINGER_DISABLE	1
public static final int	RINGER_ENABLE	2
public static final int	SELECTIVE_RECORDING	2
public static final int	UNKNOWN	4

CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv		
public static final int	ID	1073750018

CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv		
public static final int	ID	1073750025

CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv		
public static final int	ID	1073750032

CiscoAddrOutOfServiceEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv		
public static final int	ID	1073750019

CiscoAddrRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrRecordingConfigChangedEv		
public static final int	ID	1073750033

CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv		
public static final int	ID	1073750020

CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv		
public static final int	ID	1073750023

CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv		
public static final int	ID	1073741840

CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv		
public static final int	ID	1073741845

CiscoCall

com.cisco.jtapi.extensions.CiscoCall		
public static final int	CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT	2
public static final int	CALL_RECORDING_TYPE_AUTOMATIC	1
public static final int	CALL_RECORDING_TYPE_NONE	0
public static final int	CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION	4
public static final int	CALL_RECORDING_TYPE_USER_INITIATED_FROM_DEVICE	3
public static final int	CALLSECURITY_AUTHENTICATED	2
public static final int	CALLSECURITY_ENCRYPTED	3
public static final int	CALLSECURITY_NOTAUTHENTICATED	1
public static final int	CALLSECURITY_UNKNOWN	0
public static final int	CFWD_ALL_CLEAR	128
public static final int	CFWD_ALL_NONE	0
public static final int	CFWD_ALL_SET	64
public static final int	FEATUREPRIORITY_EMERGENCY	3
public static final int	FEATUREPRIORITY_NORMAL	1
public static final int	FEATUREPRIORITY_URGENT	2
public static final int	PLAYTONE_BOTHLOCALANDREMOTE	2
public static final int	PLAYTONE_LOCALONLY	0
public static final int	PLAYTONE_NOLOCAL_OR_REMOTE	3

com.cisco.jtapi.extensions.CiscoCall		
public static final int	PLAYTONE_REMOTEONLY	1
public static final int	SILENT_MONITOR	1

CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallChangedEv		
public static final int	ID	1073754120

CiscoCallCtlTermConnHeldReversionEv

com.cisco.jtapi.extensions.CiscoCallCtlTermConnHeldReversionEv		
public static final int	ID	1073762306

CiscoCallEv

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_ACCESSINFORMATIONDISCARDED	43
public static final int	CAUSE_BARGE	509
public static final int	CAUSE_BCBPRESENTLYAVAIL	58
public static final int	CAUSE_BCNAUTHORIZED	57
public static final int	CAUSE_BEARERCAPNIMPL	65
public static final int	CAUSE_CALLBEINGDELIVERED	7
public static final int	CAUSE_CALLIDINUSE	84
public static final int	CAUSE_CALLMANAGER_FAILURE	508
public static final int	CAUSE_CALLREJECTED	21
public static final int	CAUSE_CALLSPLIT	393216
public static final int	CAUSE_CHANTYPENIMPL	66
public static final int	CAUSE_CHANUNACCEPTABLE	6
public static final int	CAUSE_CTICCMSIP400BADREQUEST	16777257
public static final int	CAUSE_CTICCMSIP401UNAUTHORIZED	33554453

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_CTICCMSIP402PAYMENTREQUIRED	50331669
public static final int	CAUSE_CTICCMSIP403FORBIDDEN	67108885
public static final int	CAUSE_CTICCMSIP404NOTFOUND	83886081
public static final int	CAUSE_CTICCMSIP405METHODNOTALLOWED	100663359
public static final int	CAUSE_CTICCMSIP406NOTACCEPTABLE	117440591
public static final int	CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED	134217749
public static final int	CAUSE_CTICCMSIP408REQUESTTIMEOUT	150995046
public static final int	CAUSE_CTICCMSIP410GONE	184549398
public static final int	CAUSE_CTICCMSIP411LENGTHREQUIRED	201326719
public static final int	CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG	234881151
public static final int	CAUSE_CTICCMSIP414REQUESTURITOOLONG	251658367
public static final int	CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE	268435535
public static final int	CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME	285212799
public static final int	CAUSE_CTICCMSIP420BADEXTENSION	352321663
public static final int	CAUSE_CTICCMSIP421EXTENSTIONREQUIRED	369098879
public static final int	CAUSE_CTICCMSIP423INTERVALTOOBRIEF	402653311
public static final int	CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE	1073741842
public static final int	CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST	1090519081
public static final int	CAUSE_CTICCMSIP482LOOPDETECTED	1107296281
public static final int	CAUSE_CTICCMSIP483TOOMANYHOOPS	1124073497
public static final int	CAUSE_CTICCMSIP484ADDRESSINCOMPLETE	1140850716
public static final int	CAUSE_CTICCMSIP485AMBIGUOUS	1157627905
public static final int	CAUSE_CTICCMSIP486BUSYHERE	1174405137
public static final int	CAUSE_CTICCMSIP487REQUESTTERMINATED	1191182367
public static final int	CAUSE_CTICCMSIP488NOTACCEPTABLEHERE	1207959583
public static final int	CAUSE_CTICCMSIP491REQUESTPENDING	1258291217
public static final int	CAUSE_CTICCMSIP493UNDECIPHERABLE	1291845649
public static final int	CAUSE_CTICCMSIP500SERVERINTERNALERROR	1409286185

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_CTICCMSIP501NOTIMPLEMENTED	1426063439
public static final int	CAUSE_CTICCMSIP502BADGATEWAY	1442840614
public static final int	CAUSE_CTICCMSIP503SERVICEUNAVAILABLE	1459617833
public static final int	CAUSE_CTICCMSIP504SERVERTIMEOUT	1476395110
public static final int	CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED	1493172351
public static final int	CAUSE_CTICCMSIP513MESSAGETOOLARGE	1509949567
public static final int	CAUSE_CTICCMSIP600BUSYEVERYWHERE	-1593835503
public static final int	CAUSE_CTICCMSIP603DECLINE	-1577058283
public static final int	CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE	-1560281087
public static final int	CAUSE_CTICCMSIP606NOTACCEPTABLE	-1543503841
public static final int	CAUSE_CTICONFERENCEFULL	262144
public static final int	CAUSE_CTIDEVICENOTPREEMPTABLE	123
public static final int	CAUSE_CTIDROPCONFEREE	458752
public static final int	CAUSE_CTIMANAGER_FAILURE	507
public static final int	CAUSE_CTIPRECEDENCECALLBLOCKED	46
public static final int	CAUSE_CTIPRECEDENCELEVELEXCEEDED	122
public static final int	CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH	129
public static final int	CAUSE_CTIPREEMPTFORREUSE	9
public static final int	CAUSE_CTIPREEMPTNOREUSE	8
public static final int	CAUSE_DESTINATIONOUTOFORDER	27
public static final int	CAUSE_DESTNUMMISSANDDCNOTSUB	90
public static final int	CAUSE_DPARK	512
public static final int	CAUSE_DPARK_REMINDER	514
public static final int	CAUSE_DPARK_UNPARK	513
public static final int	CAUSE_EXCHANGEROUTINGERROR	25
public static final int	CAUSE_FAC_CMC	510
public static final int	CAUSE_FACILITYREJECTED	29
public static final int	CAUSE_IDENTIFIEDCHANDDOESNOTEXIST	82

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_IENIMPL	99
public static final int	CAUSE_INBOUNDBLINDTRANSFER	505
public static final int	CAUSE_INBOUNDCONFERENCE	504
public static final int	CAUSE_INBOUNDTRANSFER	503
public static final int	CAUSE_INCOMINGCALLBARRED	54
public static final int	CAUSE_INCOMPATABLEDESTINATION	88
public static final int	CAUSE_INTERWORKINGUNSPECIFIED	127
public static final int	CAUSE_INVALIDCALLREFVALUE	81
public static final int	CAUSE_INVALIDIECONTENTS	100
public static final int	CAUSE_INVALIDMESSAGEUNSPECIFIED	95
public static final int	CAUSE_INVALIDNUMBERFORMAT	28
public static final int	CAUSE_INVALIDTRANSITNETSEL	91
public static final int	CAUSE_MANDATORYIEMISSING	96
public static final int	CAUSE_MSGNCOMPATABLEWCS	101
public static final int	CAUSE_MSGTYPENCOMPATWCS	98
public static final int	CAUSE_MSGTYPENIMPL	97
public static final int	CAUSE_NETOUTOFORDER	38
public static final int	CAUSE_NOANSWERFROMUSER	19
public static final int	CAUSE_NOCALLSUSPENDED	85
public static final int	CAUSE_NOCIRCAVAIL	34
public static final int	CAUSE_NOERROR	0
public static final int	CAUSE_NONSELECTEDUSERCLEARING	26
public static final int	CAUSE_NORMALCALLCLEARING	16
public static final int	CAUSE_NORMALUNSPECIFIED	31
public static final int	CAUSE_NOROUTETODDESTINATION	3
public static final int	CAUSE_NOROUTETOTRANSITNET	2
public static final int	CAUSE_NOUSERRESPONDING	18
public static final int	CAUSE_NUMBERCHANGED	22

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_ONLYRDIVEARERCAPAVAIL	70
public static final int	CAUSE_OUTBOUNDCONFERENCE	502
public static final int	CAUSE_OUTBOUNDTRANSFER	501
public static final int	CAUSE_OUTOFBANDWIDTH	125
public static final int	CAUSE_PROTOCOLERRORUNSPECIFIED	111
public static final int	CAUSE_QSIG_PR	511
public static final int	CAUSE_QUALOFSERVNAVAIL	49
public static final int	CAUSE_QUIET_CLEAR	515
public static final int	CAUSE_RECOVERYONTIMEREXPIRY	102
public static final int	CAUSE_REDIRECTED	200
public static final int	CAUSE_REQCALLIDHASBEENCLEARED	86
public static final int	CAUSE_REQCIRCNAVIL	44
public static final int	CAUSE_REQFACILITYNIMPL	69
public static final int	CAUSE_REQFACILITYNOTSUBSCRIBED	50
public static final int	CAUSE_RESOURCESNAVAIL	47
public static final int	CAUSE_RESPONSETOSTATUSENQUIRY	30
public static final int	CAUSE_SERVNOTAVAILUNSPECIFIED	63
public static final int	CAUSE_SERVOPERATIONVIOLATED	53
public static final int	CAUSE_SERVOROPTNAVAILORIMPL	79
public static final int	CAUSE_SUBSCRIBERABSENT	20
public static final int	CAUSE_SUSPCALLBUTNOTTHISONE	83
public static final int	CAUSE_SWITCHINGEQUIPMENTCONGESTION	42
public static final int	CAUSE_TEMPORARYFAILURE	41
public static final int	CAUSE_UNALLOCATEDNUMBER	1
public static final int	CAUSE_USERBUSY	17

CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv		
public static final int	ID	1073754129

CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv		
public static final int	ID	1073754121

CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv		
public static final int	ID	1073754128

CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv		
public static final int	ID	1073754114

CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv		
public static final int	ID	1073754115

CiscoConnection

com.cisco.jtapi.extensions.CiscoConnection		
public static final int	ADDRESS_SEARCH_SPACE	2
public static final int	CALLED_ADDRESS_DEFAULT	0
public static final int	CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY	2
public static final int	CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION	1
public static final int	CALLED_ADDRESS_UNCHANGED	0

com.cisco.jtapi.extensions.CiscoConnection		
public static final int	CALLINGADDRESS_SEARCH_SPACE	1
public static final int	DEFAULT_SEARCH_SPACE	0
public static final int	REASON_DIRECTCALL	1
public static final int	REASON_FORWARDALL	5
public static final int	REASON_FORWARDBUSY	4
public static final int	REASON_FORWARDNOANSWER	3
public static final int	REASON_OUTBOUND	99
public static final int	REASON_REDIRECT	6
public static final int	REASON_TRANSFERREDCALL	2
public static final int	REDIRECT_DROP_ON_FAILURE	1
public static final int	REDIRECT_NORMAL	2

CiscoConnectionUniqueIDChangedEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv		
public static final int	ID	1073754116

CiscoConsultCallActiveEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv		
public static final int	ID	1073754116

CiscoFeatureReason

com.cisco.jtapi.extensions.CiscoFeatureReason		
public static final int	REASON_BARGE	20
public static final int	REASON_BLINDTRANSFER	7
public static final int	REASON_CALLPICKUP	11
public static final int	REASON_CCM_REDIRECTION	26
public static final int	REASON_CLICK_TO_CONFERENCE	39

com.cisco.jtapi.extensions.CiscoFeatureReason		
public static final int	REASON_CONFERENCE	9
public static final int	REASON_DPARK_CALLPARK	27
public static final int	REASON_DPARK_REVERSION	28
public static final int	REASON_DPARK_UNPARK	29
public static final int	REASON_FAC_CMC	22
public static final int	REASON_FORWARDALL	5
public static final int	REASON_FORWARDBUSY	4
public static final int	REASON_FORWARDNOANSWER	3
public static final int	REASON_IMMDIVERT	21
public static final int	REASON_MOBILITY	33
public static final int	REASON_MOBILITY_CELLPICKUP	35
public static final int	REASON_MOBILITY_FOLLOWME	38
public static final int	REASON_MOBILITY_HANDIN	36
public static final int	REASON_MOBILITY_HANDOUT	37
public static final int	REASON_MOBILITY_IVR	34
public static final int	REASON_NORMAL	12
public static final int	REASON_PARK	10
public static final int	REASON_PARKREMAINDER	15
public static final int	REASON_PARKREMINDER	15
public static final int	REASON_QSIG_PR	23
public static final int	REASON_REDIRECT	6
public static final int	REASON_REFER	24
public static final int	REASON_REPLACE	25
public static final int	REASON_SILENTMONITORING	31
public static final int	REASON_TRANSFER	2
public static final int	REASON_UNPARK	16

CiscoG711MediaCapability

com.cisco.jtapi.extensions.CiscoG711MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoG723MediaCapability

com.cisco.jtapi.extensions.CiscoG723MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoG729MediaCapability

com.cisco.jtapi.extensions.CiscoG729MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoGSMMediaCapability

com.cisco.jtapi.extensions.CiscoGSMMediaCapability		
public static final int	FRAMESIZE_EIGHTY_MILLISECOND_PACKET	80

CiscoJtapiException

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	ASSOCIATED_LINE_NOT_OPEN	-1932787685
public static final int	CALL_ALREADY_EXISTS	-1932787705
public static final int	CALL_DROPPED	-1932787564

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CALLHANDLE_NOTINCOMINGCALL	-1932787702
public static final int	CALLHANDLE_UNKNOWN_TO_LINECONTROL	-1932787644
public static final int	CANNOT_OPEN_DEVICE	-1932787647
public static final int	CANNOT_TERMINATE_MEDIA_ON_PHONE	-1932787690
public static final int	CFWDALL_ALREADY_SET	-1932787597
public static final int	CFWDALL_DESTN_INVALID	-1932787596
public static final int	CLUSTER_LINK_FAILURE	-1932787612
public static final int	COMMAND_NOT_IMPLEMENTED_ON_DEVICE	-1932787559
public static final int	CONFERENCE_ALREADY_PRESENT	-1932787588
public static final int	CONFERENCE_FAILED	-1932787590
public static final int	CONFERENCE_FULL	-1932787642
public static final int	CONFERENCE_INACTIVE	-1932787587
public static final int	CONFERENCE_INVALID_PARTICIPANT	-1932787589
public static final int	CTIERR_ACCESS_TO_DEVICE_DENIED	-1932787688
public static final int	CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	-1932787679
public static final int	CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	-1932787675
public static final int	CTIERR_BIB_NOT_CONFIGURED	-1932787476
public static final int	CTIERR_BIB_RESOURCE_NOT_AVAILABLE	-1932787489
public static final int	CTIERR_CALL_MANAGER_NOT_AVAILABLE	-1932787689
public static final int	CTIERR_CALL_NOT_EXISTED	-1932787533
public static final int	CTIERR_CALL_PARK_NO_DN	-1932787579
public static final int	CTIERR_CALL_REQUEST_ALREADY_OUTSTANDING	-1932787577
public static final int	CTIERR_CALL_UNPARK_FAILED	-1932787583
public static final int	CTIERR_CAPABILITIES_DO_NOT_MATCH	-1932787518
public static final int	CTIERR_CLOSE_DELAY_NOT_SUPPORTED_WITH_REG_TYPE	-1932787673
public static final int	CTIERR_CONFERENCE_ALREADY_EXISTED	-1932787535
public static final int	CTIERR_CONFERENCE_NOT_EXISTED	-1932787534
public static final int	CTIERR_CONNECTION_ON_INVALID_PORT	-1932787503

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_CONSULT_CALL_FAILURE	-1932787576
public static final int	CTIERR_CONSULTCALL_ALREADY_OUTSTANDING	-1932787640
public static final int	CTIERR_CRYPTO_CAPABILITY_MISMATCH	-1932787500
public static final int	CTIERR_CTIHANDLER_PROCESS_CREATION_FAILED	-1932787515
public static final int	CTIERR_DB_INITIALIZATION_ERROR	-1932787494
public static final int	CTIERR_DEVICE_ALREADY_OPENED	-1932787552
public static final int	CTIERR_DEVICE_ALREADY_REGISTERED_NONEXTEND	0X8CCC0127
public static final int	CTIERR_DEVICE_NOT_OPENED_YET	-1932787551
public static final int	CTIERR_DEVICE_OWNER_ALIVE_TIMER_STARTED	-1932787517
public static final int	CTIERR_DEVICE_REGISTRATION_FAILED_NOT_SUPPORTED_MEDIATYPE	-1932787490
public static final int	CTIERR_DEVICE_RESTRICTED	-1932787502
public static final int	CTIERR_DEVICE_SHUTTING_DOWN	-1932787558
public static final int	CTIERR_DIRECTORY_LOGIN_TIMEOUT	-1932787595
public static final int	CTIERR_DUPLICATE_CALL_REFERENCE	-1932787529
public static final int	CTIERR_DUPLICATE_REMOTE_DESTINATION_NUMBER	0X8CCC0122
public static final int	CTIERR_DYNREG_IPADDRMODE_MISMATCH	-1932787468
public static final int	CTIERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE	0X8CCC0126
public static final int	CTIERR_FAC_CMC_REASON_CMC_INVALID	-1932787506
public static final int	CTIERR_FAC_CMC_REASON_CMC_NEEDED	-1932787509
public static final int	CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	-1932787508
public static final int	CTIERR_FAC_CMC_REASON_FAC_INVALID	-1932787507
public static final int	CTIERR_FAC_CMC_REASON_FAC_NEEDED	-1932787510
public static final int	CTIERR_FEATURE_ALREADY_REGISTERED	-1932787575
public static final int	CTIERR_FEATURE_DATA_REJECT	-1932787565
public static final int	CTIERR_FEATURE_SELECT_FAILED	-1932787514
public static final int	CTIERR_ILLEGAL_DEVICE_TYPE	-1932787578
public static final int	CTIERR_INCOMPATIBLE_AUTOINSTALL_PROTOCOL_VERSION	-1932787629
public static final int	CTIERR_INCORRECT_MEDIA_CAPABILITY	-1932787560

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_INFORMATION_NOT_AVAILABLE	-1932787677
public static final int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_CONFIGURED	-1932787475
public static final int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_SET	-1932787493
public static final int	CTIERR_INTERCOM_SPEEDDIAL_DESTN_INVALID	-1932787492
public static final int	CTIERR_INTERCOM_TALKBACK_ALREADY_PENDING	-1932787474
public static final int	CTIERR_INTERCOM_TALKBACK_FAILURE	-1932787491
public static final int	CTIERR_INTERNAL_FAILURE	-1932787568
public static final int	CTIERR_INVALID_CALLID	-1932787487
public static final int	CTIERR_INVALID_DEVICE_NAME	-1932787678
public static final int	CTIERR_INVALID_DTMFDIGITS	-1932787561
public static final int	CTIERR_INVALID_FILTER_SIZE	-1932787625
public static final int	CTIERR_INVALID_MEDIA_DEVICE	-1932787674
public static final int	CTIERR_INVALID_MEDIA_PARAMETER	-1932787554
public static final int	CTIERR_INVALID_MEDIA_PROCESS	-1932787519
public static final int	CTIERR_INVALID_MEDIA_RESOURCE_ID	-1932787557
public static final int	CTIERR_INVALID_MESSAGE_HEADER_INFO	-1932787627
public static final int	CTIERR_INVALID_MESSAGE_LENGTH	-1932787628
public static final int	CTIERR_INVALID_MONITOR_DESTN	-1932787486
public static final int	CTIERR_INVALID_MONITOR_DN_TYPE	-1932787580
public static final int	CTIERR_INVALID_MONITORMODE	-1932787473
public static final int	CTIERR_INVALID_PARAMETER	-1932787532
public static final int	CTIERR_INVALID_PARK_DN	-1932787582
public static final int	CTIERR_INVALID_PARK_REGISTRATION_HANDLE	-1932787581
public static final int	CTIERR_INVALID_REMOTE_DESTINATION_NAME	0X8CCC0130
public static final int	CTIERR_INVALID_REMOTE_DESTINATION_NUMBER	0X8CCC0121
public static final int	CTIERR_INVALID_RESOURCE_TYPE	-1932787530
public static final int	CTIERR_IPADDRMODE_MISMATCH	-1932787469
public static final int	CTIERR_LINE_OUT_OF_SERVICE	-1932787594

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_LINE_RESTRICTED	-1932787501
public static final int	CTIERR_MAXCALL_LIMIT_REACHED	-1932787516
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC	-1932787548
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_EXTEND	0X8CCC0128
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_NONE	-1932787550
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_STATIC	-1932787549
public static final int	CTIERR_MEDIA_CAPABILITY_MISMATCH	-1932787553
public static final int	CTIERR_MEDIA_RESOURCE_NAME_SIZE_EXCEEDED	-1932787676
public static final int	CTIERR_MEDIAREGISTRATIONTYPE_DO_NOT_MATCH	-1932787567
public static final int	CTIERR_MESSAGE_TOO_BIG	-1932787626
public static final int	CTIERR_MORE_ACTIVE_CALLS_THAN_RESERVED	-1932787531
public static final int	CTIERR_NO_EXISTING_CALLS	-1932787512
public static final int	CTIERR_NO_EXISTING_CONFERENCE	-1932787527
public static final int	CTIERR_NO_RECORDING_SESSION	-1932787479
public static final int	CTIERR_NO_RESPONSE_FROM_MP	-1932787526
public static final int	CTIERR_NOT_PRESERVED_CALL	-1932787528
public static final int	CTIERR_OPERATION_FAILED_QUIETCLEAR	-1932787566
public static final int	CTIERR_OPERATION_NOT_ALLOWED	-1932787555
public static final int	CTIERR_OUT_OF_BANDWIDTH	-1932787498
public static final int	CTIERR_OWNER_NOT_ALIVE	-1932787547
public static final int	CTIERR_PENDING_ACCEPT_OR_ANSWER_REQUEST	-1932787520
public static final int	CTIERR_PENDING_START_MONITORING_REQUEST	-1932787485
public static final int	CTIERR_PENDING_START_RECORDING_REQUEST	-1932787483
public static final int	CTIERR_PENDING_STOP_RECORDING_REQUEST	-1932787482
public static final int	CTIERR_PRIMARY_CALL_INVALID	-1932787471
public static final int	CTIERR_PRIMARY_CALL_STATE_INVALID	-1932787470
public static final int	CTIERR_RECORDING_ALREADY_INPROGRESS	-1932787480
public static final int	CTIERR_RECORDING_CONFIG_NOT_MATCHING	-1932787477

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_RECORDING_SESSION_INACTIVE	-1932787478
public static final int	CTIERR_REDIRECT_UNAUTHORIZED_COMMAND_USAGE	-1932787513
public static final int	CTIERR_REGISTER_FEATURE_ACTIVATION_FAILED	-1932787585
public static final int	CTIERR_REGISTER_FEATURE_APP_ALREADY_REGISTERED	-1932787523
public static final int	CTIERR_REGISTER_FEATURE_PROVIDER_NOT_REGISTERED	-1932787524
public static final int	CTIERR_REMOTE_DEVICE_REQUEST_FAILED_ACTIVE_RD_NOT_SET	0X8CCC0124
public static final int	CTIERR_REMOTEDESTINATION_LIMIT_EXCEEDED	0X8CCC0123
public static final int	CTIERR_RESOURCE_NOT_AVAILABLE	-1932787536
public static final int	CTIERR_START_MONITORING_FAILED	-1932787484
public static final int	CTIERR_START_RECORDING_FAILED	-1932787481
public static final int	CTIERR_STATION_SHUT_DOWN	-1932787574
public static final int	CTIERR_SYSTEM_ERROR	-1932787525
public static final int	CTIERR_UDP_PASS_THROUGH_NOT_SUPPORTED	-1932787638
public static final int	CTIERR_UNKNOWN_EXCEPTION	-1932787556
public static final int	CTIERR_UNSUPPORTED_CALL_PARK_TYPE	-1932787584
public static final int	CTIERR_UNSUPPORTED_CFWD_TYPE	-1932787511
public static final int	CTIERR_USER_NOT_AUTH_FOR_SECURITY	-1932787504
public static final int	DARES_INVALID_REQ_TYPE	-1932787591
public static final int	DATA_SIZE_LIMIT_EXCEEDED	-1932787681
public static final int	DB_ERROR	-1932787691
public static final int	DB_ILLEGAL_DEVICE_TYPE	-1932787692
public static final int	DB_NO_MORE_DEVICES	-1932787694
public static final int	DESTINATION_BUSY	-1897005054
public static final int	DESTINATION_UNKNOWN	-1897005055
public static final int	DEVICE_ALREADY_REGISTERED	-1932787693
public static final int	DEVICE_NOT_OPEN	-1932787686
public static final int	DEVICE_OUT_OF_SERVICE	-1932787593
public static final int	DIGIT_GENERATION_ALREADY_IN_PROGRESS	-1932787610

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	DIGIT_GENERATION_CALLSTATE_CHANGED	-1932787607
public static final int	DIGIT_GENERATION_WRONG_CALL_HANDLE	-1932787609
public static final int	DIGIT_GENERATION_WRONG_CALL_STATE	-1932787608
public static final int	DIRECTORY_LOGIN_FAILED	-1932787616
public static final int	DIRECTORY_LOGIN_NOT_ALLOWED	-1932787617
public static final int	DIRECTORY_TEMPORARY_UNAVAILABLE	-1932787618
public static final int	EXISTING_FIRSTPARTY	-1932787709
public static final int	HOLDFAILED	-1932787697
public static final int	ILLEGAL_CALLINGPARTY	-1932787706
public static final int	ILLEGAL_CALLSTATE	-1932787703
public static final int	ILLEGAL_HANDLE	-1932787708
public static final int	ILLEGAL_MESSAGE_FORMAT	-1932787630
public static final int	INCOMPATIBLE_PROTOCOL_VERSION	-1932787632
public static final int	INVALID_LINE_HANDLE	-1932787599
public static final int	INVALID_RING_OPTION	-1932787680
public static final int	LINE_GREATER_THAN_MAX_LINE	-1932787606
public static final int	LINE_INFO_DOES_NOT_EXIST	-1932787611
public static final int	LINE_NOT_PRIMARY	-1932787598
public static final int	LINECONTROL_FAILURE	-1932787704
public static final int	MAX_NUMBER_OF_CTI_CONNECTIONS_REACHED	-1932787641
public static final int	MSGWAITING_DESTN_INVALID	-1932787592
public static final int	NO_ACTIVE_DEVICE_FOR_THIRDPARTY	-1932787710
public static final int	NO_CONFERENCE_BRIDGE	-1932787639
public static final int	NOT_INITIALIZED	-1932787613
public static final int	PROTOCOL_TIMEOUT	-1091584273
public static final int	PROVIDER_ALREADY_OPEN	-1932787614
public static final int	PROVIDER_CLOSED	-559038737
public static final int	PROVIDER_NOT_OPEN	-1932787615

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	REDIRECT_CALL_CALL_TABLE_FULL	-1932787662
public static final int	REDIRECT_CALL_DESTINATION_BUSY	-1932787649
public static final int	REDIRECT_CALL_DESTINATION_OUT_OF_ORDER	-1932787648
public static final int	REDIRECT_CALL_DIGIT_ANALYSIS_TIMEOUT	-1932787659
public static final int	REDIRECT_CALL_DOES_NOT_EXIST	-1932787683
public static final int	REDIRECT_CALL_INCOMPATIBLE_STATE	-1932787654
public static final int	REDIRECT_CALL_MEDIA_CONNECTION_FAILED	-1932787658
public static final int	REDIRECT_CALL_NORMAL_CLEARING	-1932787651
public static final int	REDIRECT_CALL_ORIGINATOR_ABANDONED	-1932787656
public static final int	REDIRECT_CALL_PARTY_TABLE_FULL	-1932787657
public static final int	REDIRECT_CALL_PENDING_REDIRECT_TRANSACTION	-1932787653
public static final int	REDIRECT_CALL_PROTOCOL_ERROR	-1932787661
public static final int	REDIRECT_CALL_UNKNOWN_DESTINATION	-1932787660
public static final int	REDIRECT_CALL_UNKNOWN_ERROR	-1932787652
public static final int	REDIRECT_CALL_UNKNOWN_PARTY	-1932787655
public static final int	REDIRECT_CALL_UNRECOGNIZED_MANAGER	-1932787650
public static final int	REDIRECT_CALLINFO_ERR	-1932787664
public static final int	REDIRECT_ERR	-1932787663
public static final int	RETRIEVEFAILED	-1932787695
public static final int	RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	-1932787600
public static final int	SSAPI_NOT_REGISTERED	-1932787684
public static final int	TIMEOUT	-1932787711
public static final int	TRANSFER_INACTIVE	-1932787586
public static final int	TRANSFERFAILED	-1932787698
public static final int	TRANSFERFAILED_CALLCONTROL_TIMEOUT	-1932787645
public static final int	TRANSFERFAILED_DESTINATION_BUSY	-1932787699
public static final int	TRANSFERFAILED_DESTINATION_UNALLOCATED	-1932787701
public static final int	TRANSFERFAILED_OUTSTANDING_TRANSFER	-1932787646

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	UNDEFINED_LINE	-1932787707
public static final int	UNKNOWN_GLOBAL_CALL_HANDLE	-1932787687
public static final int	UNRECOGNIZABLE_PDU	-1932787631
public static final int	UNSPECIFIED	0

CiscoLocales

com.cisco.jtapi.extensions.CiscoLocales		
public static final int	LOCALE_ARABIC_ALGERIA	47
public static final int	LOCALE_ARABIC_BAHRAIN	48
public static final int	LOCALE_ARABIC_EGYPT	49
public static final int	LOCALE_ARABIC_IRAQ	50
public static final int	LOCALE_ARABIC_JORDAN	51
public static final int	LOCALE_ARABIC_KUWAIT	38
public static final int	LOCALE_ARABIC_LEBANON	52
public static final int	LOCALE_ARABIC_MOROCCO	53
public static final int	LOCALE_ARABIC_OMAN	36
public static final int	LOCALE_ARABIC_QATAR	54
public static final int	LOCALE_ARABIC_SAUDI_ARABIA	37
public static final int	LOCALE_ARABIC_TUNISIA	55
public static final int	LOCALE_ARABIC_UNITED_ARAB_EMIRATES	35
public static final int	LOCALE_ARABIC_YEMEN	56
public static final int	LOCALE_BULGARIAN_BULGARIA	27
public static final int	LOCALE_CATALAN_SPAIN	32
public static final int	LOCALE_CHINESE_HONG_KONG	24
public static final int	LOCALE_CROATIAN_CROATIA	28
public static final int	LOCALE_CZECH_CZECH_REPUBLIC	26
public static final int	LOCALE_DANISH_DENMARK	12

com.cisco.jtapi.extensions.CiscoLocales		
public static final int	LOCALE_DUTCH_NETHERLAND	8
public static final int	LOCALE_ENGLISH_UNITED_KINGDOM	33
public static final int	LOCALE_ENGLISH_UNITED_STATES	1
public static final int	LOCALE_FINNISH_FINLAND	22
public static final int	LOCALE_FRENCH_FRANCE	2
public static final int	LOCALE_GERMAN_GERMANY	3
public static final int	LOCALE_GREEK_GREECE	16
public static final int	LOCALE_HEBREW_ISRAEL	39
public static final int	LOCALE_HUNGARIAN_HUNGARY	14
public static final int	LOCALE_ITALIAN_ITALY	7
public static final int	LOCALE_JAPANESE_JAPAN	13
public static final int	LOCALE_KOREAN_KOREA	21
public static final int	LOCALE_NORWEGIAN_NORWAY	9
public static final int	LOCALE_POLISH_POLAND	15
public static final int	LOCALE_PORTUGUESE_BRAZIL	23
public static final int	LOCALE_PORTUGUESE_PORTUGAL	10
public static final int	LOCALE_ROMANIAN_ROMANIA	30
public static final int	LOCALE_RUSSIAN_RUSSIA	5
public static final int	LOCALE_SERBIAN_REPUBLIC_OF_MONTENEGRO	41
public static final int	LOCALE_SERBIAN_REPUBLIC_OF_SERBIA	40
public static final int	LOCALE_SIMPLIFIED_CHINESE_CHINA	20
public static final int	LOCALE_SLOVAK_SLOVAKIA	25
public static final int	LOCALE_SLOVENIAN_SLOVENIA	29
public static final int	LOCALE_SPANISH_SPAIN	6
public static final int	LOCALE_SWEDISH_SWEDEN	11
public static final int	LOCALE_THAI_THAILAND	42
public static final int	LOCALE_TRADITIONAL_CHINESE_CHINA	19

CiscoMediaConnectionMode

com.cisco.jtapi.extensions.CiscoMediaConnectionMode		
public static final int	NONE	0
public static final int	RECEIVE_ONLY	1
public static final int	TRANSMIT_AND_RECEIVE	3
public static final int	TRANSMIT_ONLY	2

CiscoMediaEncryptionAlgorithmType

com.cisco.jtapi.extensions.CiscoMediaEncryptionAlgorithmType		
public static final int	AES_128_COUNTER	1

CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv		
public static final int	ID	1073758213

CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator		
public static final int	MEDIA_ENCRYPT_KEYS_AVAILABLE	0
public static final int	MEDIA_ENCRYPT_KEYS_UNAVAILABLE	2
public static final int	MEDIA_ENCRYPT_USER_NOT_AUTHORIZED	1
public static final int	MEDIA_NOT_ENCRYPTED	3

CiscoOutOfServiceEv

com.cisco.jtapi.extensions.CiscoOutOfServiceEv		
public static final int	CAUSE_CALLMANAGER_FAILURE	1001
public static final int	CAUSE_CTIMANAGER_FAILURE	1007
public static final int	CAUSE_DEVICE_FAILURE	1004

com.cisco.jtapi.extensions.CiscoOutOfServiceEv		
public static final int	CAUSE_DEVICE_RESTRICTED	1008
public static final int	CAUSE_DEVICE_UNREGISTERED	1005
public static final int	CAUSE_LINE_RESTRICTED	1009
public static final int	CAUSE_NOCALLMANAGER_AVAILABLE	1003
public static final int	CAUSE_REHOME_TO_HIGHER_PRIORITY_CM	1002
public static final int	CAUSE_REHOMING_FAILURE	1006
public static final int	ID	1073750021

CiscoPartyInfo

com.cisco.jtapi.extensions.CiscoPartyInfo		
public static final int	ABBREVIATED_NUMBER	6
public static final int	INTERNATIONAL_NUMBER	1
public static final int	NATIONAL_NUMBER	2
public static final int	NET_SPECIFIC_NUMBER	3
public static final int	RESERVED_FOR_EXTENSION	7
public static final int	SUBSCRIBER_NUMBER	4
public static final int	UNKNOWN_NUMBER	0

CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv		
public static final int	ID	1073754113
public static final int	PARK_STATE_ACTIVE	2
public static final int	PARK_STATE_IDLE	1
public static final int	REASON_CALLPARK	1
public static final int	REASON_CALLPARKREMAINDER	3
public static final int	REASON_CALLPARKREMINDER	3
public static final int	REASON_CALLUNPARK	2

CiscoProvFeatureID

com.cisco.jtapi.extensions.CiscoProvFeatureID		
public static final int	MONITOR_CALLPARK_DN	1234

CiscoProviderCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv		
public static final int	ID	1073741846
public static final int	MODIFY_CGPN	32
public static final int	MONITOR_PARKDN	64
public static final int	SUPERPROVIDER	16

CiscoProvTerminalCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoProvTerminalCapabilityChangedEv		
public static final int	ID	1073741847

CiscoRemoteTerminal

com.cisco.jtapi.extensions.CiscoRemoteTerminal		
public static final int	EXTEND_MEDIA_REGISTRATION	8
public static final int	NO_EXTEND_MEDIA_REGISTRATION	0

CiscoRestrictedEv

com.cisco.jtapi.extensions.CiscoRestrictedEv		
public static final int	CAUSE_UNKNOWN	0
public static final int	CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION	3
public static final int	CAUSE_UNSUPPORTED_PROTOCOL	2
public static final int	CAUSE_USER_RESTRICTED	1
public static final int	ID	1073741833

CiscoRouteSession

com.cisco.jtapi.extensions.CiscoRouteSession		
public static final int		
public static final int	CALLINGADDRESS_SEARCH_SPACE	1
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_INVALID	-1932787506
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_NEEDED	-1932787509
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	-1932787508
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_INVALID	-1932787507
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_NEEDED	-1932787510
public static final int	DEFAULT_SEARCH_SPACE	0
public static final int	DONOT_RESET_ORIGINALCALLED	0
public static final int	ERROR_INVALID_STATE	7
public static final int	ERROR_NO_CALLBACK	6
public static final int	ERROR_NONE	4
public static final int	ERROR_ROUTESELECT_TIMEOUT	5
public static final int	RESET_ORIGINALCALLED	1
public static final int	ROUTEADDRESS_SEARCH_SPACE	2

CiscoRouteTerminal

com.cisco.jtapi.extensions.CiscoRouteTerminal		
public static final int	DYNAMIC_MEDIA_REGISTRATION	2
public static final int	NO_MEDIA_REGISTRATION	0

CiscoRTPBitRate

com.cisco.jtapi.extensions.CiscoRTPBitRate		
public static final int	R5_3	1
public static final int	R6_4	2

CiscoRTPInputKeyEv

com.cisco.jtapi.extensions.CiscoRTPInputKeyEv		
public static final int	ID	1073758214

CiscoRTPInputStartedEv

com.cisco.jtapi.extensions.CiscoRTPInputStartedEv		
public static final int	ID	1073758209

CiscoRTPInputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv		
public static final int	ID	1073758210

CiscoRTPOutputKeyEv

com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv		
public static final int	ID	1073758215

CiscoRTPOutputStartedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv		
public static final int	ID	1073758211

CiscoRTPOutputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv		
public static final int	ID	1073758212

CiscoRTPPayload

com.cisco.jtapi.extensions.CiscoRTPPayload		
public static final int	ACTIVEVOICE	81

com.cisco.jtapi.extensions.CiscoRTPPayload		
public static final int	ACY_G729AASSN	15
public static final int	DATA56	33
public static final int	DATA64	32
public static final int	G711ALAW56K	3
public static final int	G711ALAW64K	2
public static final int	G711ULAW56K	5
public static final int	G711ULAW64K	4
public static final int	G722_48K	8
public static final int	G722_56K	7
public static final int	G722_64K	6
public static final int	G7231	9
public static final int	G728	10
public static final int	G729	11
public static final int	G729ANNEXA	12
public static final int	GSM	80
public static final int	IS11172AUDIOCAP	13
public static final int	IS13818AUDIOCAP	14
public static final int	NONSTANDARD	1
public static final int	WIDEBAND_256K	25

CiscoTermActivatedEv

com.cisco.jtapi.extensions.CiscoTermActivatedEv		
public static final int	ID	1073741843

CiscoTermButtonPressedEv

com.cisco.jtapi.extensions.CiscoTermButtonPressedEv		
public static final int	CHARA	10

com.cisco.jtapi.extensions.CiscoTermButtonPressedEv		
public static final int	CHARB	11
public static final int	CHARC	12
public static final int	CHARD	13
public static final int	EIGHT	8
public static final int	FIVE	5
public static final int	FOUR	4
public static final int	ID	1073745936
public static final int	NINE	9
public static final int	ONE	1
public static final int	POUND	15
public static final int	SEVEN	7
public static final int	SIX	6
public static final int	STAR	14
public static final int	THREE	3
public static final int	TWO	2
public static final int	ZERO	0

CiscoTermConnMonitoringEndEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringEndEv		
public static final int	ID	1073762311

CiscoTermConnMonitoringStartEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringStartEv		
public static final int	ID	1073762310

CiscoTermConnMonitorInitiatorInfoEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorInitiatorInfoEv		
public static final int	ID	1073762313

CiscoTermConnMonitorTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorTargetInfoEv		
public static final int	ID	1073762314

CiscoTermConnPrivacyChangedEv

com.cisco.jtapi.extensions.CiscoTermConnPrivacyChangedEv		
public static final int	ID	1073762305

CiscoTermConnRecordingEndEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingEndEv		
public static final int	ID	1073762309

CiscoTermConnRecordingStartEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingStartEv		
public static final int	ID	1073762308

CiscoTermConnRecordingTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingTargetInfoEv		
public static final int	ID	1073762312

CiscoTermConnSelectChangedEv

com.cisco.jtapi.extensions.CiscoTermConnSelectChangedEv		
public static final int	ID	1073762307

CiscoTermCreatedEv

com.cisco.jtapi.extensions.CiscoTermCreatedEv		
public static final int	ID	1073745921

CiscoTermDataEv

com.cisco.jtapi.extensions.CiscoTermDataEv		
public static final int	ID	1073745922

CiscoTermDeviceStateActiveEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv		
public static final int	ID	1073745926

CiscoTermDeviceStateAlertingEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv		
public static final int	ID	1073745927

CiscoTermDeviceStateHeldEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv		
public static final int	ID	1073745928

CiscoTermDeviceStateIdleEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv		
public static final int	ID	1073745929

CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv		
public static final int	ID	1073745941

CiscoTermDNDOptionChangedEv

com.cisco.jtapi.extensions.CiscoTermDNDOptionChangedEv		
public static final int	ID	1073745942

CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv		
public static final int	ID	1073745940

CiscoTerminal

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	ASCII_ENCODING	2
public static final int	DEVICESTATE_ACTIVE	1
public static final int	DEVICESTATE_ALERTING	2
public static final int	DEVICESTATE_HELD	3
public static final int	DEVICESTATE_IDLE	0
public static final int	DEVICESTATE_UNKNOWN	4
public static final int	DEVICESTATE_WHISPER	5
public static final int	CiscoTerminal.DEVICETYPE_UNKNOWN	0
public static final int	CiscoTerminal.DEVICETYPE_ANALOG_PHONE	30027
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6901	547
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6911	548
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6921	495
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6941	496
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6945	564
public static final int	CiscoTerminal.DEVICETYPE_CISCO_6961	497
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7906	369
public static final int	CiscoTerminal.DEVICETYPE_TELECASTER_BID	6
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7911	307

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	CiscoTerminal.DEVICETYPE_14_BUTTON_SIDECAR	124
public static final int	CiscoTerminal.DEVICETYPE_7915_12_BUTTON_SIDECAR	227
public static final int	CiscoTerminal.DEVICETYPE_7915_24_BUTTON_SIDECAR	228
public static final int	CiscoTerminal.DEVICETYPE_7916_12_BUTTON_SIDECAR	229
public static final int	CiscoTerminal.DEVICETYPE_7916_24_BUTTON_SIDECAR	230
public static final int	CiscoTerminal.DEVICETYPE_CKEM_36_BUTTON	232
public static final int	CiscoTerminal.DEVICETYPE_CP7921	365
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7925	484
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7926	577
public static final int	CiscoTerminal.DEVICETYPE_7931	348
public static final int	CiscoTerminal.DEVICETYPE_IP_CONFERENCE_PHONE	9
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7936	30019
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7937	431
public static final int	CiscoTerminal.DEVICETYPE_TELECASTER_BUSINESS	8
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7941	115
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7941G_GE	309
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7942	434
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7945	435
public static final int	CiscoTerminal.DEVICETYPE_TELECASTER_MGR	7
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7961	30018
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7961G_GE	308
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7962	404
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7965	436
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7970	30006
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7971	119
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7975	437
public static final int	CiscoTerminal.DEVICETYPE_CISCO_7989	302
public static final int	CiscoTerminal.DEVICETYPE_CISCO_8941	586

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	CiscoTerminal.DEVICETYPE_CISCO_8945	585
public static final int	CiscoTerminal.DEVICETYPE_CISCO_8961	540
public static final int	CiscoTerminal.DEVICETYPE_9951	537
public static final int	CiscoTerminal.DEVICETYPE_CISCO_9971	493
public static final int	CiscoTerminal.DEVICETYPE_ATA_186	12
public static final int	CiscoTerminal.DEVICETYPE_CISCO_ATA_187	550
public static final int	CiscoTerminal.DEVICETYPE_CISCO_CIUS	593
public static final int	CiscoTerminal.DEVICETYPE_CISCO_CIUS_SP	632
public static final int	CiscoTerminal.DEVICETYPE_CISCO_SOFTPHONE_SE_M	30016
public static final int	CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_COMMUNICATOR	358
public static final int	CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_MOBILE_COMMUNICATOR	468
public static final int	CiscoTerminal.DEVICETYPE_CISCO_UNIFIED_COMMUNICATIONS_FOR_RTX	648
public static final int	CiscoTerminal.DEVICETYPE_CLIENT_SERVICES_FRAMEWORK	503
public static final int	CiscoTerminal.DEVICETYPE_VGC_PHONE	10
public static final int	CiscoTerminal.DEVICETYPE_CTI_PORT	72
public static final int	CiscoTerminal.DEVICETYPE_CTI_ROUTE_POINT	73
public static final int	CiscoTerminal.DEVICETYPE_DEVICE_PILOT	71
public static final int	CiscoTerminal.DEVICETYPE_ISDN_BRI_PHONE	30028
public static final int	CiscoTerminal.DEVICETYPE_CTI_REMOTE_DEVICE	635
public static final int	DND_OPTION_CALL_REJECT	2
public static final int	DND_OPTION_NONE	0
public static final int	DND_OPTION_RINGER_OFF	1
public static final int	IN_SERVICE	1
public static final int	IP_ADDRESSING_MODE_IPV4	0
public static final int	IP_ADDRESSING_MODE_IPV4_V6	2
public static final int	IP_ADDRESSING_MODE_IPV6	1
public static final int	IP_ADDRESSING_MODE_UNKNOWN	3
public static final int	IP_ADDRESSING_MODE_UNKNOWN_ANATRED	4

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	NOT_APPLICABLE	1
public static final int	OUT_OF_SERVICE	0
public static final int	UCS2UNICODE_ENCODING	3
public static final int	UNKNOWN_ENCODING	0

CiscoTerminalConnection

com.cisco.jtapi.extensions.CiscoTerminalConnection		
public static final int	CISCO_SELECTEDLOCAL	1
public static final int	CISCO_SELECTEDNONE	0
public static final int	CISCO_SELECTEDREMOTE	2
public static final int	PROTOCOL_CTI_REMOTE_DEVICE	3
public static final int	RECORDING_INVOCATION_TYPE_SILENT	0
public static final int	RECORDING_INVOCATION_TYPE_USER	1

CiscoTerminalProtocol

com.cisco.jtapi.extensions.CiscoTerminalProtocol		
public static final int	PROTOCOL_NONE	0
public static final int	PROTOCOL_SCCP	1
public static final int	PROTOCOL_SIP	2

CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv		
public static final int	ID	1073745923

CiscoTermOutOfServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv		
public static final int	ID	1073745924

CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv		
public static final int	DB_INITIALIZATION_ERROR	15
public static final int	ID	1073745937
public static final int	IP_ADDRESSING_MODE_MISMATCH	19
public static final int	MEDIA_ALREADY_TERMINATED_DYNAMIC	8
public static final int	MEDIA_ALREADY_TERMINATED_NONE	6
public static final int	MEDIA_ALREADY_TERMINATED_STATIC	7
public static final int	MEDIA_CAPABILITY_MISMATCH	10
public static final int	OWNER_NOT_ALIVE	11
public static final int	UNKNOWN	0

CiscoTermRemovedEv

com.cisco.jtapi.extensions.CiscoTermRemovedEv		
public static final int	ID	1073745925

CiscoTermRestrictedEv

com.cisco.jtapi.extensions.CiscoTermRestrictedEv		
public static final int	ID	1073741841

CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv		
public static final int	ID	1073745939

CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv		
public static final int	ID	1073745938

CiscoTone

com.cisco.jtapi.extensions.CiscoTone		
public static final int	ZIPZIP	49

CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoToneChangedEv		
public static final int	CMC_REQUIRED	1
public static final int	FAC_CMC_REQUIRED	2
public static final int	FAC_REQUIRED	0
public static final int	ID	1073754119

CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferEndEv		
public static final int	ID	1073754117

CiscoTransferStartEv

com.cisco.jtapi.extensions.CiscoTransferStartEv		
public static final int	ID	1073754118

CiscoUrlInfo

com.cisco.jtapi.extensions.CiscoUrlInfo		
public static final int	TRANSPORT_TYPE_TCP	1
public static final int	TRANSPORT_TYPE_UDP	2

com.cisco.jtapi.extensions.CiscoUrlInfo		
public static final int	URL_TYPE_SIP	1
public static final int	URL_TYPE_TEL	2
public static final int	URL_TYPE_UNKNOWN	0

CiscoWideBandMediaCapability

com.cisco.jtapi.extensions.CiscoWideBandMediaCapability		
public static final int	FRAMESIZE_TEN_MILLISECOND_PACKET	10

Alarm

com.cisco.services.alarm.Alarm		
public static final int	ALERTS	1
public static final int	CRITICAL	2
public static final int	DEBUGGING	7
public static final int	EMERGENCIES	0
public static final int	ERROR	3
public static final int	HIGHEST_LEVEL	7
public static final int	INFORMATIONAL	6
public static final int	LOWEST_LEVEL	0
public static final int	NO_SEVERITY	-1
public static final int	NOTIFICATION	5
public static final java.lang.String	UNKNOWN_MNEMONIC	“UNK”
public static final int	WARNING	4

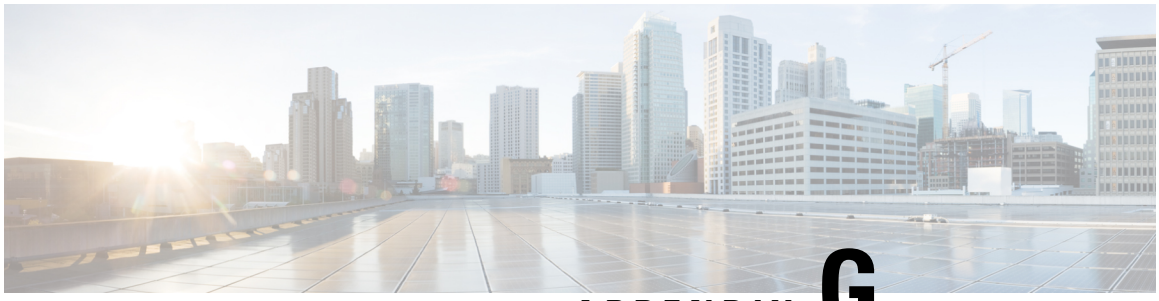
LogFileTraceWriter

com.cisco.services.tracing.LogFileTraceWriter		
public static final java.lang.String	DEFAULT_FILE_NAME_BASE	“trace”

com.cisco.services.tracing.LogFileTraceWriter		
public static final java.lang.String	DEFAULT_FILE_NAME_EXTENSION	“log”
public static final char	DIR_BASE_NAME_NUM_SEPERATOR	95
public static final int	MIN_FILE_SIZE	10240
public static final int	MIN_FILES	2
public static final int	ROLLOVER_THRESHOLD	1024

Trace

com.cisco.services.tracing.Trace		
public static final int	ALERTS	1
public static final java.lang.String	ALERTS_TRACE_NAME	“ALERTS”
public static final int	CRITICAL	2
public static final java.lang.String	CRITICAL_TRACE_NAME	“CRITICAL”
public static final int	DEBUGGING	7
public static final java.lang.String	DEBUGGING_TRACE_NAME	“DEBUGGING”
public static final int	EMERGENCIES	0
public static final java.lang.String	EMERGENCIES_TRACE_NAME	“EMERGENCIES”
public static final int	ERROR	3
public static final java.lang.String	ERROR_TRACE_NAME	“ERROR”
public static final int	HIGHEST_LEVEL	7
public static final int	INFORMATIONAL	6
public static final java.lang.String	INFORMATIONAL_TRACE_NAME	“INFORMATIONAL”
public static final int	LOWEST_LEVEL	0
public static final int	NOTIFICATION	5
public static final java.lang.String	NOTIFICATION_TRACE_NAME	“NOTIFICATION”
public static final int	WARNING	4
public static final java.lang.String	WARNING_TRACE_NAME	“WARNING”



APPENDIX G

Caveats

This appendix provides details of the JTAPI caveats that are common across releases and those that are release-specific.

Caveats for All Releases

This section lists the caveats that are common for all JTAPI releases and contains these topics:

• Unsupported CTI Events for SIP Phones, on page 1706

Single Versus Multiple CallObserver Clarification

There are two primary ways to observe addresses with Cisco JTAPI's CallObservers: an application can observe all of the addresses with a single CallObserver object or it can have a separate CallObserver object for each of address. These two approaches cause slightly different events to be seen, mostly with regard to reason codes.

When an application uses a single CallObserver to observe all the addresses, they are connected with one object. When A calls B, both the events at A and at B are sent to the same CallObserver object. If B redirects to C, and C was observed with the same object, all its events are delivered to the same observer. The application observes the CallCtlConnOfferedEv to C with reason REDIRECT, because the observer at C knew all about the previous events on the call.

Conversely, when an application uses an independent CallObserver for each Address, this information is not so easily shared. When A calls B, call events of A go to the observer for A, and B's go to the observer for B. They each know about the other end, for example A will know that B is ringing, but they are no longer the same observer. When B redirects the call to C, the observer at C knows absolutely nothing about the call. The observer at C was not involved in the original call at all, and does not know who is on it, what events had happened previously. This information has to be made up by JTAPI to build an accurate call model at C. All the call events for a basic call between A and B have to be simulated so that the call model, from C's perspective makes sense.

This is done by using a snapshot event. JTAPI looks at the call, in this case the one between A and B, and figures out what events have to have happened for the call to exist the way it does. This makes up up the basic call events required, and give them to the observer on C, so that it can build a proper call model.

Since this event set is made up by JTAPI, the reason codes are not available. For example, if A had originally called D, and D redirected to B, the made up snapshot event set would not be concerned with the redirect at all. JTAPI does not store this information anywhere, and when it generates a snapshot, it creates the simplest event set possible to recreate the call model, and reports all the events with reason NORMAL.

So, when A calls B, and then B redirects to C, the observer on C gets a snapshot event that allows it to recreate the call model for a basic call from A to B. Also in this snapshot event is the CallCtlConnOfferedEv for C. As part of the snapshot, this event comes in with reason NORMAL, even though it is the result of a redirect. CallObservers on A or B will see the CallCtlConnOfferedEv for C with reason REDIRECT, but there is no way for the observer at C to know that.

This creates a noticeable difference in the reason codes available to applications depending on how they implement their CallObservers. There have not been any issues regarding this from the customer side.

This is the way it has been since Cisco JTAPI's inception and this is a clarification of the existing behavior.

SIP and SCCP Dialing Differences with Overlapping Directory Number Patterns

An overlapping Directory Number Pattern is when one Directory Number is a part of a longer Directory Number. For example, a Directory Number 1000 overlaps a Directory Number 10001. Cisco Unified Communications Manager (Cisco Unified CM) and JTAPI both support overlapping Directory Number patterns, but there are some important things to note regarding this.

When you dial 1000 from a normal phone, there is a delay. The Cisco Unified CM does not know if you want to dial 1000 or 10001, so it waits for you to make a choice. This Digit Analyzer waits for 15 seconds if you press nothing else. During this time, nothing happens, no call is extended, and it just sounds like a dead call

for the calling party. This can be short circuited by hitting # to let Cisco Unified CM know that you actually intend to call 1000. The 15 second wait is a configurable service parameter, known as T302 Timer, and can be set as low as 3 seconds.

JTAPI using SCCP phones avoid the Digit Analyzer entirely by communicating directly with Cisco Unified CM. JTAPI sends the intended Directory Number when it is making a call, and if it sends 1000, it means only that, and Cisco Unified CM knows it will not be dialing any more digits.

JTAPI using SIP phones is quite different. JTAPI communicates with the phone, which then communicates with the Cisco Unified CM. The phone takes care of the dialing, and JTAPI will pass it the digits 1000 to dial. Due to this, JTAPI cannot avoid the Digit Analyzer, and is subject to the T302 wait outlined above. JTAPI sits idly and waits while the Digit Analyzer figures out that the SIP phone actually wants to dial 1000.

Apart from this, by default the JTAPI CTI Postcondition value is also set to 15 seconds. This means that when JTAPI sends a request to the CTI layer, it waits for 15 seconds before it assumes something has gone wrong, and throws a timeout exception. This means that the delay for Digit Analysis for overlapping DN patterns is very likely to cause JTAPI to time out.

The Digit Analysis delay cannot be completely removed for SIP phones, but this problem can be greatly mitigated through the use of service or jtapi.ini parameters. As noted above, the T302 Timer for Digit Analysis can be set as low as three seconds, which is much lower than the 15 it takes JTAPI to time out. You can also increase the JTAPI CTI Postcondition timeout to 20 seconds in the jtapi.ini file. This issue can also be avoided by not having overlapping DNs.

Translation Pattern Support

If the callingparty transformation mask is configured for a Translation Pattern that is applied to the controlled addresses of JTAPI application, the application may observe some extra connections being created and disconnected when the application observes both calling and called party. Otherwise, a connection is created for the transformed callingparty and CiscoCall.getCurrentCallingParty() returns the transformed calling party address when the application observes only the called party. In general, JTAPI has a problem in creating an appropriate call connection and may not be able to provide correct callinfo such as currentCalling, currentCalled, calling, called, and lastRedirecting parties. For example, Translation Pattern X is configured with calling party transformation mask Y and calledparty transformation mask B; A calls X, and call goes to B. In this scenario:

- If the application observes only B, JTAPI creates connection for Y and B, and CiscoCall.getCurrentCallingParty() returns Address Y.
- If the application observes both A and B, connections for A and B are created, connection for Y is temporarily created and dropped, and CiscoCall.getCurrentCallingParty() returns Address Y.

Other inconsistencies could occur in callinfo if more features are used for basic call. It is recommended that you do not configure callingparty transformation mask for Translation Pattern which might get applied to JTAPI Application controlled Addresses.

DT24+ Limitation with PRI NI2 Trunk

When a PRI NI2 trunk used by DT24+ gateway is involved in a call scenario between two clusters, for example, A from cluster-1 calls B in cluster-2 via DT24+ PRI NI2 trunk, the LastRedirectAddress and CalledAddress may not be accurate on B’s side. Besides, if there are any changes for A’s side of the call in cluster-1 due to redirect, transfer, or forward, the changed information is not propagated to B’s side due to protocol limitation of PRI NI2 truck.

Connection for Park Number Not Created

JTAPI does not create a Queued state connection for Park Number if the call is parked across the gateway. There are two possibilities here:

1. If CLI is configured, application sees an Unknown connection
2. If CLI is not configured, the calling does not see any Unknown connection.

For Example, If A calls B across gateway (with CLI configured) and B parks the call then A sees an unknown connection instead of a connection (with STATE = QUEUED) for Park Number.

But, if A calls B across gateway (with no CLI configured) and B parks the call then A does not see any new connection.

Inconsistency Between SIP and SCCP Phone

sendData() API on CiscoTerminal is used to send data to the phone. In case of SIP phones, if invalid byte data is sent by the application, the method throws PlatformException. However, in case of SCCP phones, the byte data sent in the request is not validated, and would return successfully without throwing an exception.

Failure to Route Calls Across Destinations

When a call is redirected to a device outside the cluster over an H323 gateway that is out of service, before call control can determine the Out-of-Service status of H323 gateway, the call is disconnected. This is because the default value of the service parameter CTI NewCallAccept timer is four seconds where as call control takes five seconds to determine that the gateway is out of service, so calls are disconnected due to expiration of CTI NewCallAccept timeout.

Implications of the above behavior is seen in JTAPI selectRoute() API which internally uses CTI redirect API to route the call. If applications specify multiple destinations with selectRoute() and the first destination is across an out-of-service H323 gateway, the call fails before JTAPI can route the call to the second destination. Hence JTAPI, cannot route the call to the second route specified in selectRoute() interface call.

To avoid this, the value of CTI New Call Accept Timer service parameter can be set as greater than H225 TCP Timer service parameter.

Incorrect Return Value for getCallingAddress()

In a transfer scenario, where caller and transferController are not observed, that is, where A calls B and transfers the call to C and the application is observing only C then before the transfer, JTAPI will not have any information about the first call (that is, call from A to B). So, when the transfer feature is invoked, the calling and called address are B and C respectively. On completing the transfer, application updates callInfo and JTAPI exposes the correct parties through getCurrentCallingAddress(), getCurrentCalledAddress(), getModifiedCallingAddress(), and getModifiedCalledAddress(). However, getCallingAddress() API which should return the original calling address still reports B, that is, the original calling party of B to C call.

To avoid this issue, application can observe the controller as well, so that JTAPI expose the correct party (that is A, in this case) with getCallingAddress() API.

Call Fails to Disconnect Held Shared Line

In a scenario where A calls B (B is a shared line present on terminals T1 and T2); privacy is set as ON for T1 and initially CUCM service parameter Enforce Privacy Settings on held calls is set to True. B(T1) answers and goes to Talking state while B(T2) goes to Passive state (TermConn = Passive; CallCtlTermConn = InUse). B(T1) puts the call on hold, since Enforce Privacy Setting on held calls is set to True, B(T2) remains in passive state (TermConn = Passive; CallCtlTermConn = InUse). Now the service parameter Enforce Privacy Settings on held calls is set to False. This does not trigger any change in the state of TerminalConnection, so B(T2) still remains Passive-InUse (TermConn = Passive; CallCtlTermConn = InUse). At this point, if the application sets the requestController as B(T1) and disconnects the call at B, the connection of B is not disconnected and call does not go IDLE. Even on the phones, the call on A remains in Established state while the other party in call is B(T2) which remains in Passive-InUse (TermConn = Passive; CallCtlTermConn = InUse) state. Call is cleared when A disconnects the call.

Limitation with sendData() API on CiscoTerminal

If JTAPI applications make simultaneous back to back requests for sendData() API on the same CiscoTerminal, without any delay between requests, then some of these requests may fail. Applications cannot determine whether a request was successful or not, as Cisco JTAPI API returns successfully as soon as the phone receives data and does not wait for a response from the phone. Also, the IP phone might display a blank screen on sending simultaneous requests to send data.

To avoid these issues, JTAPI applications should ensure some time delay between two successive sendData() requests while pushing XSI data to the IP phones via Cisco JTAPI.

Limitation in Using ; (Semi-Colon) and = (Equal) in User ID and Password

Sun JTAPI 1.2 specification does not support use of the semicolon ';' and equals ' = ' characters when populating the Host Name, UserID, and Password fields in string used as parameter in getProvider() method. If ';' or ' = ' are used in these fields, items such as 'pass = word' or 'pass;word' are treated as 'pass' and your request could fail and you must not use these characters in userid and password fields.

Connection to Unknown Address When Unparking a Conference Call

When a conference call is parked, JTAPI call will have connection to the remaining parties in the call. When this call is unparked using the UnPark API or connect API, connections to unknown address will be temporarily created. This connection to unknown address will be disconnected when un park is completed.

The following will be seen in JTAPI traces:

2002 : (P1-InProv) 103023/1 ConnCreatedEv Unknown:null:4 187 Cause:100 CallCtlCause:0 CiscoCause:31 FeatReason:10 CAUSE_NORMAL

....

2002 : (P1-InProv) 103023/1 ConnDisconnectedEv Unknown:null:4 187 Cause:100 CallCtlCause:0 CiscoCause:31 FeatReason:10 CAUSE_NORMAL

CTI Redirect to Voice Mail Wont Work with QSIG

When applications redirect a call to voice mail through a QSIG trunk, voice mail will not play the voice prompt of the redirecting party. A generic prompt asking the user to enter the voice mail box is played. This is due to the fact that CTI redirect doesn't pass correct original called party and redirecting party to the voice mail system. Applications can work around this by using a SIP trunk.

Phone A calls Phone B (CUCILYNC in Deskphone mode). Toast popup of "envelope" appears for incoming call.

User clicks on the envelope, redirecting the call across an H.323 trunk with QSIG tunneling. Unity/Unity Connection won't recognize original called party, so call will not go to user's voice mailbox. This is because Re-direct/Original Called Party information not carried across the trunk when CTI redirect is used.

CiscoAddress.getForwarding() Returns Correct Value Only for In-Service Addresses

Although the Sun JTAPI 1.2 specification does not specify any preconditions for CallControlAddress.getForwarding(), the implementation of this method in Cisco JTAPI will return a correct value only if the address is in service. When invoked on an out of service address this method will just return a NULL value.

Unsupported CTI Events for SIP Phones

The following CTI events are not generated for SIP phones. Third party applications that expect these call events should use SCCP phones:

- CallOpenLogicalChannelEvent
- CallRingEvent
- DeviceLampModeChangedEvent
- DeviceModeChangedEvent
- DeviceDisplayChangedEvent
- DeviceFeatureButtonPressedEvent
- DeviceKeyPressedEvent
- DeviceLampModeChangedEvent
- DeviceRingModeChangedEvent

Caveats for Release 9.1(1)

This section lists the JTAPI caveats for Release 9.1(1).

- Connection for Park DN While UnPark, on page 1707

Connection for Park DN While UnPark

Cisco JTAPI will no longer create the temporary connections for the Park/DPark number in the final call when a parked call is unparked.

This change in behavior will be seen in and above the following Cisco JTAPI versions - 8.5.1.10000-10, 8.6.1.10000-4, 8.6.2.10000-2 and 8.6.2.98000-2

Scenario:

GC1: A calls B, B parks a call is parked at 1000

GC2: B unparks

“Preserve globalCallId for Parked Calls” is set to false

Previous behavior on unpark:

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 ConnCreatedEv 1000
GC2 CallCtlConnEstablishedEv B
GC2 ConnCreatedEv A
GC2 ConnDisconnetedEv 1000
GC2 CallCtlConnEstablishedEv A
GC1 CallChangedEv
GC1 ConnDisconenctedEv 1000
GC1 ConnDisconenctedEv A
GC1 CallInvalidEv
```

Current behavior on unpark: Connection for 1000 will not be created in GC2

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 CallCtlConnEstablishedEv B
GC2 ConnCreatedEv A
GC2 CallCtlConnEstablishedEv A
GC1 CallChangedEv
GC1 ConnDisconenctedEv 1000
GC1 ConnDisconenctedEv A
GC1 CallInvalidEv
```

"Preserve globalCallId for Parked Calls" is set to true

Previous behavior on unpark:

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 ConnCreatedEv 1000
GC2 CallCtlConnEstablishedEv B
Gc2 CallChangedEv
GC2 ConnDisconnetedEv 1000
GC2 CallCtlConnDisconnectedEv 1000
GC2 ConnDisconnetedEv B
GC2 CallCtlConnDisconnectedEv B
GC2 CallInvalidEv
GC1 ConnCreatedEv 1000
GC1 ConnCreatedEv B
Gc2 ConnDisconnectedEv 1000
GC1 CallCtlConnEstablishedEv B
```

Current behavior on unpark: Connection for 1000 will not be created in GC1

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 CallCtlConnEstablishedEv B
GC1 ConnDisconnectedEv 1000
GC1 CallCtlConnDisconnectedEv 1000
Gc2 CallChangedEv
GC1 ConnCreatedEv B
GC1 ConnConnectedEv B
GC1 CallCtlCallEstablishedEv B
GC2 ConnDisconnetedEv B
GC2 CallCtlConnDisconnectedEv B
GC2 CallInvalidEv
```

Caveats for Release 8.6(1)

This section lists the JTAPI caveats for Release 8.6(1).

• Limitation While Using a Cisco Telepresense MCU, on page 1708

Limitation While Using a Cisco Telepresense MCU

In scenarios, where a conference chaining is done across cluster, the number of connection seen by the application might not be correct and application may not see the connection for the external conference bridge getting created.

Example:

```
A,  B and B' are in Cluster 1D is in cluster 2
Application is observing D
GC1: A calls D; Call offered on D and D answers
GC2: D consults B; B answers
B' CBarges into the call
D completes conference - GC1.conference(GC2)
```

After the conference is complete the application will see only the connection for A and D but not that of the conference bridge.

Caveats for Release 8.5(1)

This section lists the JTAPI caveats for Release 8.5(1).

• Discouraged Use of JTAPIProperties.updateCertificate(), on page 1708
• Delete SecurityProperties Before Re-Use, on page 1709
• No ConnDisconnectedEv Event When Call Is Rejected, on page 1709

Discouraged Use of JTAPIProperties.updateCertificate()

Applications are encouraged to move away from using the JTAPIProperties.updateCertificate() method to download certificates. The JTAPIProperties.setSecurityPropertyForInstance() method is superior for most applications, as it will store the security information in the jtapi.ini file, giving all the information to JTAPI

whenever the application chooses to connect securely later. This information is critical for JTAPI to use the correct settings to create a secure connection, and if an application chooses to use the JTAPIProperties.updateCertificate() method, then the information will not be stored in the jtapi.ini file.

Applications that have no reason not to, should move to setSecurityPropertyForInstance().

If an application is intentionally using the updateCertificate() method to avoid the use of jtapi.ini configurations, then the application must provide the information in the JTAPI Provider String, when it is first initializing JTAPI.

The required fields are:

- InstanceID
- CertStorePassphrase
- CAPF
- CAPFPort
- TFTP
- TFTPPort
- CertPath

If an application chooses to use the updateCertificate() method and chooses not to provide the required security information in the Provider String, the behavior of JTAPI is not guaranteed.

Delete SecurityProperties Before Re-Use

The JTAPIProperties.setSecurityPropertyForInstance() method downloads certificates to disk, and stores security information related to them in the jtapi.ini file. If an application is interested in downloading new certificates to disk, changing the security information for the certificates, or both, is it recommended that they invoke JTAPIProperties.deleteSecurityPropertyForInstance() before doing so. This method will delete the certificates from disk, and remove the related SecurityProperty from the jtapi.ini file.

This will ensure a fresh start for the new set of certificates, and help to eliminate any errors that could arise from "stale" certificates lingering around on disk.

No ConnDisconnectedEv Event When Call Is Rejected

For CUCM version 8.5.1 and later versions, when a call is made from A (observed) to B (unobserved) from application and if call is rejected for any reason, application will get the ConnFailedEv/CallCtlConnFailedEv events for the calling party, but there might be just one connection created in JTAPI for the calling party and there might not be any ConnDisconnectedEv/CallCtlConnDisconnectedEv events for called party. In addition, the ConnDisconnectedEv/CallCtlConnDisconnectedEv events for the calling party would be generated only after the calling party device/phone clears the call which itself can take 30 secs or more. Therefore, if application does not want to wait for the Disconnect events, upon getting the Failed events, it can simply clear the call directly after catching the failure from its call.connect() request as 'PlatformExceptionImpl caught: Could not meet post conditions of connect()'.

Caveats for Release 8.0(1)

This section lists the JTAPI caveats for Release 8.0(1).

Globalized Calling Party Number

This caveat is a further clarification for Globalized Calling Party Number. With 8.0(1), there have been changes to the Call Processing and CTI layers that reflect the globalized calling party values that you see on the station more accurately. There are still some serious limitations with globalized parties:

- The Globalized Calling Party Number is available on the called side.
- The Globalized Calling Party Number is determined only when the call is offered. This applies to basic calls and for calls involving features.
    - The Globalized Calling Party Number does not change, even if the calling party changes. This is a clarification for the existing caveat Globalized Calling Party Number.
- Globalized Calling Party will be applicable if the call is redirected, whether performed prior to call being connected or after call is connected.
- Globalized Calling Party will not be provided after the these features are completed: Transfer, Conference, Unpark, Auto Call Pickup.

Conference Interaction with Chaperone Results in Unsupported Conference Chaining

If both caller and Chaperone complete conference, where caller completes conference before chaperone then the scenario results in Conference Chaining. As of release 8.0(1), such scenarios are not supported by JTAPI and currently connections for all the parties along with ConferenceChain connection are shown in a single call.

For example, A calls B; call is intercepted by Chaperone (C1) which further consults B for conference. Before Chaperone completes conference, Caller(A) goes ahead and consults X for conference and completes the conference. After this, Chaperone(C1) completes the conference. This scenario would result in unexpected behavior from JTAPI Perspective. JTAPI could send CiscoConferenceStartedEv along with CiscoChainAddedEv. Also, after the conference, JTAPI shows connections for two Conference chains, A(caller), B, X, and C1(Chaperone) in the same call.

This will be fixed in future releases by having connections of caller and X in one call which is chained to a different call with connections for C1 and B. The fix requires changes in multiple components which are tracked with CDETS: CSCtc76213(CTI), CSCtc76222(TSP), CSCtc76223(Conference)

Wildcard Routepoint Interaction

Before 8.0(1), Wildcard Routepoint scenarios were not supported by JTAPI. This behavior is maintained in this release, in the spirit of Backward Compatibility, but is still unsupported. A new Service Parameter has been introduced in 8.0(1), WildCardDN as Called Party, which fixes several issues in the call model for Wildcard Routepoint scenarios. When this service parameter is turned on, Wildcard Routepoints are fully supported by JTAPI.

Inconsistent Address Type of ModifiedCalledAddress When a Call Is Made to a Hunt Pilot

When a call is made to a Hunt Pilot, although the API call.getCurrentCallingAddress() will return an address of type CiscoAddress.HUNT_PILOT, the address type of the address returned by the API call.getModifiedCalledAddress() will not be CiscoAddress.HUNT_PILOT.

This is because the modified address, as the name suggests, might be modified by application and getType() on modified address would return CiscoAddress.UNKNOWN or CiscoAddress.INTERNAL and JTAPI currently has no way to determine if that corresponds to a Hunt Pilot or not.

Caveats for Release 7.0.1

This section lists the JTAPI caveats for Release 7.0.1.

Inconsistency in getModifiedCallingAddress()

When a call is made to a shared DN (Address shared) on two Terminals (A and B), configured with different Calling Party Transformation CSS, and try to transform the Calling Party differently through two different Calling Party Transformation Patterns, then JTAPI does not provide modifiedCallingParty consistently. In such a scenario, both devices send localization signals to call control to transform the calling party number and the one picked by call control first is used to transform the same and JTAPI returns that through getModifiedCallingAddress().

For Example, +918055552222 (Globalized Calling Party) calls JTAPI observed shared Line 3100 on Terminals A and B. Device A is configured to transform the calling party by removing International escape character where as, B is configured to transform the calling party by removing International Escape Character and country code 91. Then JTAPI cannot guarantee whether getModifiedCallingAddress(), that is, the Localized Calling Party, will return 918055552222 or 8055552222.

Conference Behavior for Selected and Active Calls

Following is the behavior when application issues conference request but selected and active calls are not part of the conference request:

Active Call on a Terminal is always added to the resulting conference when conference is invoked on a call on any address on that terminal.

Consider B1 and B2 are addresses on same terminal

A --> B1- GC1

C --> B1- GC2

D --> B2- GC3 (active call)

The application invokes GC1.conference(GC2) and results in A-B1-C-D in conference with GC1, although call with D was not part of the conference request.

Active conference call on a terminal is added to the resulting conference when conference is invoked on a call on any line on that terminal. In this case, the active conference call becomes the surviving final call (provided the application specified primary call is not a conference call). In this scenario, the application specified primary call is cleared after the conference. It is possible that the application specified primary call may not join the resulting conference and in that case the call is not cleared after conference is over.

Consider B1 and B2 are addresses on the same terminal and conf1 is a conference call with A-B1-C in conference with B1 as the controller

B1 --> D - GC1 (on hold)

conf1 - GC2 (active call)

B2 --> E - GC3 (on hold)

The application invokes GC1.conference(GC2, GC3).

This results in A-B1-C-D-E in conference with GC2 as the surviving call. Although application had specified GC1 to be the primary call, GC1 does not survive after the conference.

The same behavior applies for user selected calls that are not part of the conference request, but become part of the resulting conference as mentioned above.

The same behavior would apply to a regular conference with common controller. Consider A, B, C, D as lines on different terminals

A-->B - GC1

C-->B - GC2

D-->B - GC3 (active call)

The application requests GC1.conference(GC2). This results in A-B-C-D in conference with GC1. Although direct call with D was not part of the conference request, D will join the conference

Change in GlobalizedCallingParty Behavior

getGlobalizedCallingParty() returns the correct globalized calling number only in case of a basic call. In a scenario where A calls B, B answers. A and B are connected. In this case, if application requests for getGlobalizedCallingParty(), the API returns the globalized number for A. In case of features such as transfer or redirect where any of the party gets updated, getGlobalizedCallingParty() may not return the correct globalized number.

Caveats for Release 6.0.1

This section lists the JTAPI caveats for Release 6.0.1:

Call History Might Get Lost When AAR Routes Over QSIG Trunk

When a call is forwarded due to insufficient bandwidth (Call Forward No Bandwidth - CFNB) to another cluster over a trunk or gateway using QSIG, call history might get lost. If Phone A calls Phone B, which is in a low bandwidth location, with CFNB set to forward calls to Phone C, which is in a different cluster, and the QSIG protocol is used on the trunk/gateway, then the original called party and the last redirecting party might not get passed to the destination party.

Different Event Order If Consult Call Initiated on SIP Device

Scenario: Terminal A initiates a call to the shared line B/B' and which Initiates a consult call to Terminal C.

- If the Shared Line is SIP device then the call events are :
  - B (active) receives: **CallCtlTermConnHeldEv** > **CiscoTermConnSelectChangedEv** > **CallActive**
  - B' (remote-in-use) receives: **CiscoTermConnSelectChangedEv** > **CallActive** > **CallCtlTermConnHeldEv**
- If the Shared Line is SCCP device then the call events are:
  - **CiscoTermConnSelectChangedEv** > **CallCtlTermConnHeldEv** > **CallActive** on both the Terminals.

Caveats for Release 5.0

This section lists the JTAPI caveats for Release 5.0:

SRTP Support

The default sRTP policy used by IPPhones is different from the one published (standard) as part of libSRTP code. libSRTP code is a free download, available at http://srtp.sourceforge.net/download.html. If the correct srtp policy is not used, the end result is no audio at both ends.

The srtp_policy is used by media terminating endpoint to create a crypto context. It should match to encrypt and decrypt packets sent/received by IPPhones/CTIPorts. Phone is using one hardcoded srtp_policy for all phone types including sip phones.

```
policy->cipher_type  =  AES_128_ICM;policy->cipher_key_len  =  30;
policy->auth_type  =  HMAC_SHA1;
policy->auth_key_len  =  20;
policy->auth_tag_len  =  4 ; // changed to 4 from 10;
policy->sec_serv  =  sec_serv_conf_and_auth;
```

JTAPI clients doing their own media termination with SRTP, must use the above policy to create a crypto context. The default policy published as part of libSRTP (the standard policy documented as part of RFC) is not same as used by Cisco Unified IP Phone. Phones use the modified one to optimize the bandwidth. The sRTP policy must be part of negotiation between the endpoints but right now only one is supported and ccm does not support the negotiation, hence applications need to use the above policy to terminate media.

Partition Support

When same Directory Number with different partitions exist in a CallManager, getAddress(String number) API in JTAPI normally returns the first matching Address object which has the same Directory Number. This is done to maintain BWC in case of no partitions configured in the system.

TLS Security

When client certificate is installed, server certificates are also downloaded from CallManager TFTP server. However, server certificate is only verified for currect signature and server trust is not established. Hence it is recomented that first time download of certificate is done in trusted network.

CiscoFeatureReason

When a call redirectred across a GW is offered at an address, getCiscoFeatureReason() returns REASON_FORWARDNOANSWER.

Within the same cluster when a call is redirected, the redirect destination sees REASON_REDIRECT as the CiscoFeatureReason. However, when a call is redirected across a cluster, through a Q.931 trunk, at the redirect destination getCiscoFeatureReason() returns REASON_FORWARDNOANSWER(as per Q.931 standard).

Scenario: A, B and C are registered to 3 clusters. A calls B, B answers and redirects the call to C. When call is offered at C, getCiscoFeatureReason() will return REASON_FORWARDNOANSWER.

Unicode Issue in Calls Involving SIP Trunks

In SIP call scenarios, where the call comes back to the call manager from the SIP proxy over a SIP trunk, since the call manager is totally dependent on the SIP, messages to populate any names and since the SIP protocol has no way of populating both ASCII and Unicode, the passed-in name is just ASCII and the same

gets sent to JTAPI. Hence, in such call scenarios, the user will not be able to see Unicode names since this is not under the control of JTAPI.

As a result the following APIs on CiscoCall interface will return only ASCII values instead of Unicode in such scenarios.

```
public String getCurrentCalledPartyUnicodeDisplayName();public String
getCurrentCallingPartyUnicodeDisplayName();
```

For SIP phone specific issues please refer to differences in SIP support section.

Join Across Lines: Conference Two or More Addresses on Same Terminal

For Unified Communications Manager releases 6.x and 5.x, if applications try to conference two or more addresses on the same terminal, based on the order of participants in the request, application may receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT for the conference request and later the conference may be created successfully with some of the participants. In such a case, there is no guarantee which one of them joins the conference. But the conference is created with only one of the address on a terminal and others are ignored. This depends on how this feature request is processed in different CUCM releases.

Below are the details of two scenarios affected:

Note This issue has been resolved in 7.x using CSCsj06488 and CSCsj06533.

1. Consider B1 and B2 are different address on the same terminal TB.

   A ->B1 - GC1

   A ->B2 - GC2

   A ->C - GC3

   Application issues a conference request GC1.conference(GC2, GC3)

   In 5.x and 6.x, application will receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT, however A, B1, C come into conference, and the normal set of events delivered in a conference scenario are seen like mentioned below:

   GC1 CiscoConferenceStartEv

   GC2 TermConnDroppedEv TB

   GC2 CallCtlTermConnDroppedEv TB

   GC2 ConnDisconnectedEv B1

   GC2 CallCtlConnDisconnectedEv B1

   GC1 CallCtlTermConnTalkingEv TB

   GC2 CiscoCallChangedEv

   GC1 ConnCreatedEv C

   GC1 ConnConnectedEv C

GC1 CallCtlConnEstablishedEv C

GC1 TermConnCreatedEv TC

GC1 TermConnActiveEv TC

GC1 CallCtlTermConnTalkingEv TC

GC2 TermConnDroppedEv TC

GC2 CallCtlTermConnDroppedEv TC

GC2 ConnDisconnectedEv C

GC2 CallCtlConnDisconnectedEv C

GC2 CallInvalidEv

GC1 CiscoConferenceEndEv

2. Consider B1 and B2 are different address on the same terminal TB.

   A ->B1 - GC1

   A ->B2 - GC2

   A ->C - GC3

   Application issues a conference request GC3.conference(GC1, GC2)

   In 5.x, Application will not receive any exception and the request would be procressed successfully. A, C, B1 would be in conference along with the regular set of conference events.

   In 6.x, Application will receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT, however A, C, B1 come into conference, and the normal set of events delivered in a conference

JTAPI Exposes Incorrect Information with getCallingAddress() and getCalledAddress()

In some scenarios where feature works on multiple calls (pickup, transfer, conference and so on) depending on Event Order or the parties observed, JTAPI may end up reporting incorrect call information to the applications. These scenarios are ones where surviving call is initially not there in JTAPI's provider domain or in scenarios where SurvivingCall goes invalid and is recreated in the middle of the feature operation. For such survivingCalls, JTAPI does not report correct information with getCallingAddress() and getCalledAddress(). Some of these scenarios are:

1. Transfer - A calls B; B transfers the call to C and application is observing only C
2. HuntList Transfer - In 8.x release if transfer is done, then surviving call can go invalid and recreated if caller is not observed.
3. PickUp scenario where survivingCall goes invalid and is recreated in middle of the feature
4. UnPark scenarios where caller is not observed and service parameter, Preserve globalCallId for Parked Calls, is set to true.

In general, this issue can happen with scenario where survivingCall is not in provider's domain initially or if is there and goes Invalid(CallInvalidEv is sent) and is recreated(CallActiveEv is sent) during the feature operation.

Caveats for Release 4.1

This section lists the JTAPI caveats for Release 4.1:

FAC-CMC

1. Forwarding should not be configured to a DN that requires FAC-CMC code. Forwarding requests will be successful, but calls will not be forwarded to these DNs and will be rejected.
2. Application should always terminate the code with #, otherwise system waits for T302 timer before extending the call. For these cases, application could get postConditionTimeOut exception for call.connect() or call.consult() but call may actually be offered. If apps need to avoid this, either all the digits with # terminated string are entered with post condition timeout (which is by default 15 sec in JTAPI Prefs UI) in the PlatformException or increase the postcondition timeout.
3. Two identical CiscoToneChangedEvents are sent to applications and second one needs to be ignored if both the codes are entered with # separated upon receiving the first event.

setConferenceController

The party that starts a conference by adding a new party acts as the original conference controller. Only the original conference controller can add new parties into the conference. If the original conference controller drops out of the conference, no other party in that particular conference call can add a new party. Although the conference controller cannot be changed while a conference call is going on, applications can determine which TerminalConnection acts as the conference controller when initially setting up a conference call via the CallControlCall.setConferenceController() method. The CallControlCall.getConferenceController() method returns the current conference controller, or null if there is none. If no conference controller is set, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked."

Consider the following scenario as an example:

A, B, C, and D belong to a conference call and all are in the TALKING state. A acts as the conference controller. A attempts to use the SetConferenceController API to change the conference controller to B, gets no error, and drops out of the conference. B then tries to add a new party, E, into the conference but cannot do so.

conference(Call[])

Applications can control which TerminalConnection acts as the conference controller when setting up a conference call via the CallControlCall.setConferenceController() method. The CallControlCall.getConferenceController() method returns the current conference controller, or null if there is none. If no conference controller is set initially, the implementation chooses a suitable TerminalConnection

when the conferencing feature is invoked. Only the original conference controller can add new parties to a conference call. Attempting to change the conference controller while a conference is going on will not take effect; however, no error gets thrown in the setConferenceController API

Interval During DTMF Digits

As per fix for CSCef05359, Change PlayDTMF to allow applications specify the time delay, now applications can configure the time delay during DTMF digits through Admin page, Service pameter, generate DTMF delay, for call Manager.

Shared Lines Support

Cisco JTAPI does not support configuration of same Directory Number from different partitions on the same or any device but configuration of different Directory Number from different partitions on the same device as well as different devices is supported.

CP Requires Previous Calls on the Device to Be in Connected Call State

CP requires previous calls on the device to be in connected call state before answering further calls on the same device. If calls are answered without checking for the call state of previous calls on the same device, then CTI might return a successful answer response but the call will not go to connected state and needs to be answered again. See DDTS CSCee17001 for more details.

CallInfo for Calls on QSIG Trunk

Call info on a call across a QSIG gateway is not consistent. Due to the limitations in the protocol, application would see inconsistent values for call.getLastRedirectingAddress() and call.getCalledAddress() for calls across QSIG trunks.

Refer to CSCee74730, CSCee59084, CSCee70747 and CSCsk62441 for details.

Caveats for 4.0

This section lists the JTAPI caveats for Release 4.0:

Extra Connection with Wild Card DN

JTAPI may create extra connection when Call is made wild card DN. See the release note of DDTS CSCeb57849 for scenarios.

CallInfo in Barge Scenario

CallInfo is not updated when Barge Initiator Drops the Call. See the release note of DDTS CSCec23359 for scenarios.

CallInfo Issues When Caller Redirects Call

In this scenario A calls B, A redirects to C and application is monitoring only C, the calling and called address would be B and C and the calling terminal would be terminal A.

Translation Pattern and Presentation Indication Interaction

JTAPI has getCalledAddressPI and getCurrentCalledAddressPi interfaces to receive the PIs of the originalCalled and Called parties. While making a call through a Translation Pattern, if the pattern modifies the PIs of the CalledParty, the getCalledAddressPI continues to reflect the earlier PI while the getCurrentCalledAddressPI shows the PI as set at the pattern. Refer DDTS CSCec58085 for more details.

Extra TermConnHeld Events

Applications may some times see an extra TermConnHeld Event on Controller during Transfer and Conference scenario. For transfer scenario this extra TermConnHeldEvent is sent on controller before TermConnDropped is sent. For conference scenario, for primary controller which remains in the call, this event is sent before TermConnTalking is sent, and for controller which is dropped from conference, it is sent before TermConnDropped is sent. Refer DDTS CSCec55257 for more details.

Transfer and Conference Interaction

- 9.1.6.1—in JTAPI whenever transfer is done to connect a normal call to a ConferenceCall, the GlobalCallId of Conference Call always survives irrespective of how the transfer is performed (whether on Call1 or Call2).
- 9.1.6.2—during transferring of conference scenario, it is possible External Connection creation events are delivered before CiscoTransferStartedEv, however, Call Merge events are still delivered within CiscoTransferStart and CiscoTransferEnd boundry.
- 9.1.6.3—during transferring of conference scenario, if transfer-destination is not observed by the JTAPI Application, the LastRedirectingParty is not updated. For example, if A, B, C are in conference, C (Transfer controller) transfers call to D, and D (Transfer-destination) is not observed by JTAPI, then

LastRedirectingParty of the call remains unchanged after Transfer is completed. Ideally after Transfer is completed, LastRedirectingParty is updated to Transfer controller.

Dropping a Call on Shared Lines

If there is heldCall on SharedAddress (SharedLine) and application is not observing all the terminals of SharedLine, then Connection.disconnect() using SharedAddress's connection does not drop the call. The is left with connections of OtherParty in the call. To drop the call, the application must use either Call.drop() or manually disconnect call from non observered terminals. In other words, if there is HeldCall on SharedAddress, then for the terminals that is not in Application's control, callmust be dropped manually. For detais, refer DDTS CSCed06910.

Barge Call

In a Barge Call, when

1. Barge target holds
2. Barge target does a consult conference or arbitrary Conference
3. The OtherParty drops the call
4. Barge target initiate transfer, arbitrary transfer or BlindTransfer
5. Barge target parks the call
6. Barge target Idiverts the call
7. Barge target redirects call

Then, Initiators TerminalConnection/CallCtlTerminalConnection is dropped as result of above operations. However, CallCtlCause for these TermConnDropp/CallCtlTermConnDrop would be Cause_Normal. JTAPI is unable to provide specific cause such as Cause_Redirect for #7 above.

Note Initiator is the party that presses the Barge key to barge into a call. Target is where Initiator Barges, OtherParty is address which not Initiator/Target. For details, refer DDTS CSCed07230.

Null lastRedirectingAddress

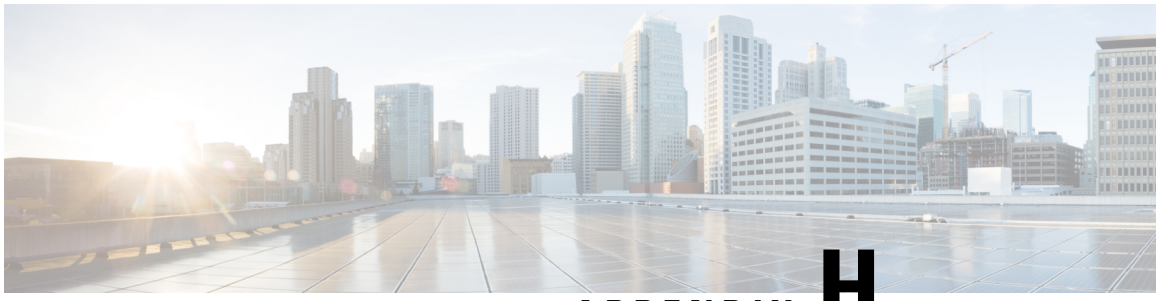
Call.getLastRedirectingAdress() returns null when only the redirecting address is observed. The call goes to INVALID state after redirect request and if application calls the above interface after redirecting the call it would see null. For details, refer CSCee92111.

Devices Configured with Same CLI

In a conference scenario, where conference controller is observed by JTAPI and other two calls are from devices configured with same the CLI (Directory Number), JTAPI creates one connection. So, when conference is completed, it has only two connections in the call instead of three. If the conference is completed using conference() API, a post condition timeout exception is thrown. For details, refer DDTS CSCeh05723.

Current Called Address

CiscoCall.getCurrentCalledAddress() returns null before the call is offered to the called address. In earlier releases, this used to return Unknown address. Applications must handle null or Unknown address returned for CiscoCall.getCurrentCalledAddress().



APPENDIX H

Deprecated API

This appendix lists lists the APIs, fields, and methods that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Deprecated Interfaces

com.cisco.jtapi.extensions.CiscoRouteAddress

This interface has not been implemented.

Deprecated Fields

Deprecated fields
com.cisco.jtapi.extensions.CiscoAddress.APPLICATION_CONTROLLED_RECORDING com.cisco.jtapi.extensions.CiscoAddress.DEVICE_CONTROLLED_RECORDING These constants are deprecated. Applications that upgrade to Release 9.0 or later releases should use the new SELECTIVE_RECORDING constant and not the deprecated APPLICATION_CONTROLLED_RECORDING and DEVICE_CONTROLLED_RECORDING constants. In Release 9.0 or later releases Unified CM and JTAPI never return the DEVICE_CONTROLLED_RECORDING constant.
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.MODIFY_CGPN This constant is not returned by any interface, should not be used by application.
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.MONITOR_PARKDN This constant is not returned by any interface, should not be used by application.
com.cisco.jtapi.extensions.CiscoProvCallParkEv.REASON_CALLPARKREMAINDER This interface is deprecated due to a spelling error. Use the new interface REASON_CALLPARKREMINDER.

Deprecated fields
com.cisco.jtapi.extensions.CiscoFeatureReason.REASON_PARKREMAINDER Use REASON_PARKREMINDER.
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.SUPERPROVIDER This constant is not returned by any interface, should not be used by application.

Deprecated Methods

Deprecated methods
com.cisco.jtapi.extensions.CiscoTermDataEv.getData() Use byte[] getTermData
com.cisco.jtapi.extensions.CiscoJtapiException.getErrorDescription(int) Use String getErrorDescription (); instead.
com.cisco.jtapi.extensions.CiscoJtapiException.getErrorName(int) Use String getErrorName (); instead.
com.cisco.jtapi.extensions.CiscoConsultCallActiveEv.getHeldTerminalConnection() Replaced by CiscoConsultCall.getConsultingTerminalConnection()
com.cisco.jtapi.extensions.CiscoCall.getLastRedirectingPartyInfo() - use getLastRedirectedPartyInfo();
com.cisco.jtapi.extensions.CiscoAddress.getRegistrationState() This method has been replaced by the getState() method.
com.cisco.jtapi.extensions.CiscoTerminal.getRegistrationState() This method has been replaced by the getState() method.
com.cisco.jtapi.extensions.CiscoMediaTerminal.register(InetAddress, int)
com.cisco.jtapi.extensions.CiscoTerminal.sendData(String)
com.cisco.jtapi.extensions.CiscoJtapiProperties.setSecurityPropertyForInstance(String, String, String, String, String, String, String, String, boolean) This method is replace by overloaded method setSecurityPropertyForInstance which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.
com.cisco.services.tracing.TraceManager.setSubFacilities(String[]) and replaced with TraceManager.addSubFacilities method

Deprecated methods
com.cisco.services.tracing.implementation.TraceManagerImpl.setSubFacilities(String[]) replaced by addSubFacilties(String[])
com.cisco.services.tracing.TraceManager.setSubFacility(String) and replaced with TraceManager.addSubFacility method
com.cisco.services.tracing.implementation.TraceManagerImpl.setSubFacility(String) replaced by addSubFacility(String)
com.cisco.jtapi.extensions.CiscoJtapiProperties.updateCertificate(String, String, String, String, String, String, String, String) This method is replace by overloaded method updateCertifcate which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.
com.cisco.jtapi.extensions.CiscoJtapiProperties.updateServerCertificate(String, String, String, String, String) This method is replace by overloaded method updateServerCertifcate which takes an extra parameter certStorePassphrase, a passphrase for java key store. This method might have some security vulnerability.

Deprecated API