# 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 Also see CTI Supported Devices and Cisco Unified JTAPI Operations by Release 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.

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.

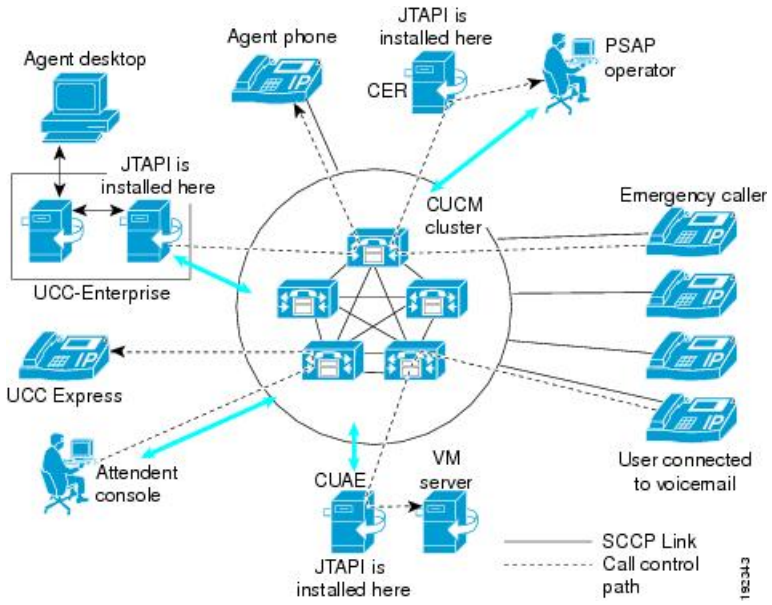
## Cisco Unified JTAPI and Enterprises

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
- Fields in the jtapi.ini File

# 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:

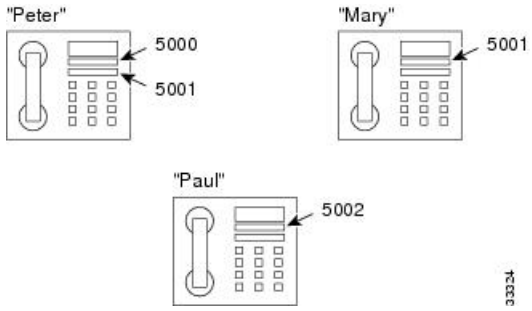
- Phones
- Virtual devices (media termination points and route points)
- Gateways

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.

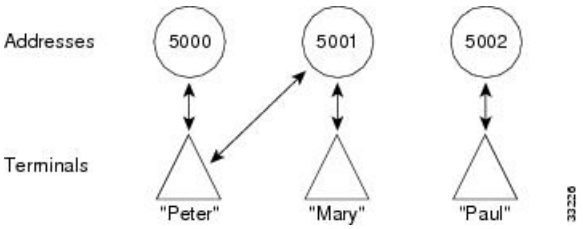
***Figure 2: Phone Diagram***




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()`<br>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()`<br>Returns an array of terminals on which this address is restricted. If none are restricted, this method returns null.<br>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)`<br>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()`<br>Returns the terminal that is activated and is removed from the restricted list. |

### CiscoOutOfServiceEv

| | |
|---|---|
| static int | `CAUSE_DEVICE_RESTRICTED`<br>Indicates whether an event is sent because a device is restricted. |
| static int | `CAUSE_LINE_RESTRICTED`<br>Indicates whether an event is sent because a line is restricted. |

### CiscoCallEv

| | |
|---|---|
| static int | `CAUSE_DEVICE_RESTRICTED`<br>Indicates whether an event is sent because a device is restricted. |
| static int | `CAUSE_LINE_RESTRICTED`<br>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.