Features Supported by Cisco Unified JTAPI

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

- Account Lockout, on page 5
- Agent Greeting, on page 5
- AES 256 Algorithm IDs, on page 6
- Alternate Script Support, on page 7
- API for Exposing Built-In-Bridge Status, on page 7
- Arabic and Hebrew Language Support, on page 8
- Auto Updater for Linux, on page 8
- AutoAccept Support for CTI Ports and Route Points, on page 9
- Autoupdate of API, on page 10
- Barge and Privacy Event Notification, on page 12
- Call Control Discovery, on page 13
- Call Forward, on page 13
- Call Forward Override, on page 13
- Call Park, on page 14
- Call Pickup, on page 14
- Call Recording for SIP or TLS Authenticated Calls, on page 15
- Call Select Status, on page 15
- Calling Party Display Name, on page 15
- Calling Party IP Address, on page 16
- Calling Party IP Address, on page 16
- Calling Party Normalization, on page 17
- CallFwdAll Key Press Notification, on page 18
- CallSelect and UnSelect Event Notification, on page 18
- Certificate Download API Enhancement, on page 19
- Changes in DeviceType Name Handling, on page 19
- Cisco MediaTerminal, on page 20
- Cisco Unified Communications Manager Media Endpoint Model, on page 22
- Cisco Unified Communications Manager Server Failure, on page 24
- Cisco Unified IP 7931G Phone Interaction, on page 25
- Cisco Unified JTAPI Install Internationalization, on page 26
- Cisco VG248 and ATA 186 Analog Phone Gateways, on page 26
- CiscoJtapiExceptions, on page 27

- CiscoProvAuthenticationInfoEv, on page 28
- CiscoRTPHandle Interface on Cisco RTP Events, on page 28
- Cisco Terminal Filter and ButtonPressedEvents, on page 29
- CiscoTermRegistrationfailed Event, on page 30
- Cius Persistency, on page 31
- Clear Calls, on page 32
- Click to Conference, on page 32
- Cluster Abstraction, on page 33
- Command Line Invocation, on page 34
- Component Updater, on page 34
- Conference, on page 34
- Conference and Join, on page 37
- Conference Chaining, on page 39
- Consult Without Media, on page 40
- CTI Ports, on page 40
- CTI RoutePoints, on page 40
- CTI Remote Device for JTAPI, on page 41
- CTI RD Call Forward, on page 44
- CTI Video Support, on page 45
- Default CTI IP Addressing for Devices, on page 46
- DeleteCall, on page 47
- Device Recovery, on page 47
- Device Recovery for Phones, on page 47
- Device State Server, on page 47
- Direct Transfer Across Lines, on page 49
- Directed Call Park, on page 54
- Directory Change Notification, on page 55
- Do Not Disturb, on page 55
- Do Not Disturb-Reject, on page 56
- Drop Any Party, on page 57
- Dynamic CTI Port Registration, on page 58
- E911 Teleworker, on page 60
- Enable or Disable Ringer, on page 60
- Encryption Enhancement, on page 61
- End to End Call Tracing, on page 61
- EnergyWise Deep Sleep Mode, on page 62
- Extension Mobility Cross Cluster, on page 64
- Extension Mobility Username Login, on page 65
- External Call Control, on page 65
- End to End Session ID for Calls, on page 66
- FIPS Compliance, on page 67
- Forced Authorization and Client Matter Codes, on page 70
- Forwarding on No Bandwidth and Unregistered DN, on page 72
- GetCallID in RTP Events, on page 72
- GetCallInfo, on page 73
- GetGlobalCallID, on page 73

- Hairpin Support, on page 73
- Half-Duplex Media Support, on page 74
- Hold Reversion, on page 75
- Hunt List, on page 75
- Hunt List Connected Number, on page 77
- Hunt Log Status, on page 77
- Intercom, on page 78
- Intercom Support for Extension Mobility, on page 80
- IPv6 Support, on page 80
- iSac Codec, on page 81
- Java Socket Connect Timeout, on page 82
- Join Across Lines, on page 82
- Join Across Lines (Only SCCP), on page 83
- Join Across Lines with Conference Enhancements (SCCP and SIP), on page 88
- JRE 1.2 and JRE 1.3 Support Removal, on page 89
- JTAPI Version Information, on page 90
- Locale Infrastructure Development, on page 90
- Logical Partitioning, on page 91
- Media Termination at Route Point, on page 91
- Media Termination Extensions, on page 94
- Message Waiting Indicator Enhancement, on page 94
- Modifying Calling Number, on page 95
- Multi-fork Recording using CUBE Media Proxy Server, on page 97
- Multilevel Precedence and Preemption Support, on page 97
- Multiple Calls Per DN, on page 97
- Native Queuing, on page 97
- Network Alerting, on page 99
- Network Events, on page 100
- New Error Code in CiscoTermRegistrationFailedEv, on page 100
- Noncontroller Adding of Parties to Conferences, on page 101
- Park DN Monitor, on page 101
- Park Monitoring and Assisted DPark Support, on page 101
- Park Reminder, on page 103
- Park Retrieval, on page 103
- Partition Support, on page 104
- Password Expiry, on page 107
- Persistent Connection, on page 107
- Play Zip Tone, on page 109
- Presentation Indicator for Calls, on page 110
- Privacy On Hold, on page 111
- Progress State Converted to Disconnect State, on page 112
- Q.Signaling (QSIG) Path Replacement, on page 112
- QoS Support, on page 112
- Quiet Clear, on page 114
- Receiving and Responding to Media Flow Events, on page 114
- Recording, on page 116

- Redirect, on page 119
- Redirect Set Original Called ID, on page 120
- Redirect to Device, on page 121
- Redundancy, on page 122
- Redundancy in CTI Managers, on page 122
- Ringback on SIP 183 for Transferred Calls, on page 125
- Routing, on page 125
- Secure Conferencing, on page 127
- Secure Real-Time Protocol Key Material, on page 128
- Secured Monitoring and Recording, on page 134
- SelectRoute Interface Enhancement, on page 135
- selectRoute() with Calling Search Space and Feature Priority, on page 136
- Set MessageWaiting, on page 136
- Shared Line Support, on page 137
- Silent Monitoring, on page 139
- Single Sign-On, on page 142
- Single Step Transfer, on page 143
- SIP 3XX Redirection, on page 144
- SIP Phone Support, on page 145
- SIP REFER or REPLACE, on page 148
- SIP Trunk Early Offer, on page 149
- Star (*) 50 Update, on page 152
- Super Provider (Disable Device Validation), on page 152
- Superprovider and Change Notification, on page 153
- Support for Cisco Unified IP Phone 6901, on page 155
- Support for Cisco Unified IP Phone 6900 Series, on page 156
- Support for 100+ Directory Numbers, on page 157
- Support for VMware, on page 158
- Swap or Cancel and Transfer or Conference Behavior, on page 159
- Terminal and Address Capability Settings, on page 160
- Terminal and Address Restrictions, on page 161
- SHA-512 Support for Digital Signatures, on page 165
- Transfer, on page 165
- Transfer and Conference Extensions, on page 168
- Transfer and DirectTransfer, on page 168
- Translation Pattern Support, on page 169
- Transport Layer Security (TLS), on page 170
- Unicode Support, on page 177
- Unrestricted Unified CM, on page 179
- URI Dialing, on page 180
- Version Format Change, on page 181
- Verification Involving PSTN Reachability, on page 181
- Video Capabilities and Multi-Media Information, on page 181
- Video On Hold Support, on page 185
- Voice MailBox Support, on page 186
- XSI Object Pass Through, on page 186

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 27

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, CiscoFeatureReason, CiscoJtapiException, CiscoMediaStreamStartedEv, CiscoMediaStreamEndedEv

Message Sequences

See Agent Greeting

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

Message Sequences

See API for Exposing Built-in-Bridge Status

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 To view the message flow for AutoAccept on CTIPort and RoutePoint, see Message Sequence Charts

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 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 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 To view the message flow for barge, CBarge, and privacy, see Message Sequence Charts.

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

Message Sequences

See Call Control Discovery

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, CiscoAddress, CiscoTerminal, CiscoProvider, CiscoProviderCapabilities, CiscoProvPickupCallAlertEv, ProviderPickupNotificationRegistrationClosedEv, CiscoAddrPickupGroupChangedEv.

Message Sequences

Call Pickup

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 and CiscoRouteEvent.

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 and CiscoPartyInfo for more information.

Message sequences

See Calling Party Normalization

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

Message Sequences

See CallFwdAll Keys Press Notification

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

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.

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 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 27.

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
- CiscoRTPInputStoppedEv
- CiscoRTPOutputStartedEv
- CiscoRTPOutputStoppedEv

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 To view the message flow for CiscoTerminal Filter and ButtonPressedEvents, see Message Sequence Charts

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 for more information.

Message Sequences

See Cius Persistency.

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

Message Sequences

Click to Conference

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 1: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box, on page 33 illustrates "Figure 1: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box, on page 33." Figure 2: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client, on page 33 illustrates "Figure 2: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client, on page 33."

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

Figure 1: Single-Box Configuration with JTAPI, Cisco Unified Communications Manager, and CTIManager in One Box

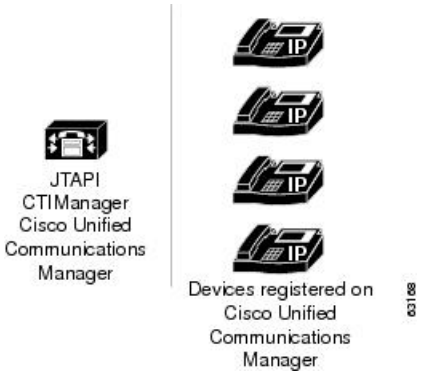
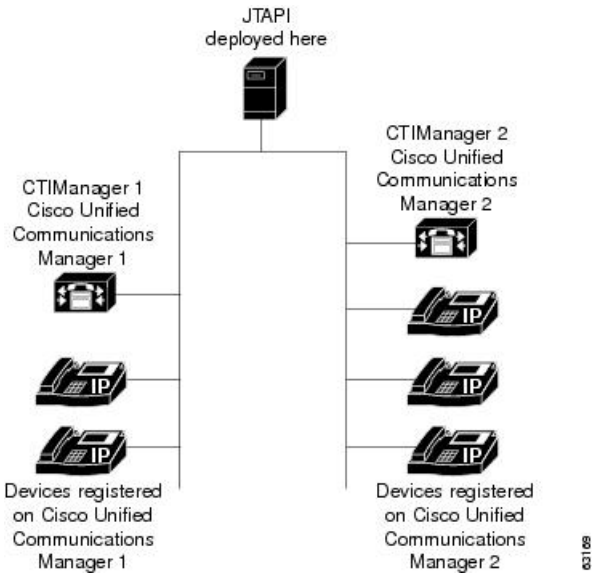


Figure 2: Redundant Cisco Unified Communications Manager and CTIManagers with JTAPI Deployed as a Separate Client



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

Message Sequences

See ComponentUpdater Enhancement Use Cases

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 Event Cause	Call	Event	Fields
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.

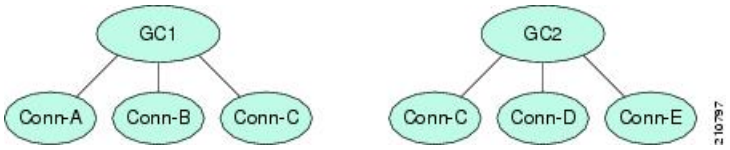
For details on the interface changes, see Cisco Unified JTAPI Extensions To view the message flow for conference and join, see Message Sequence Charts

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.

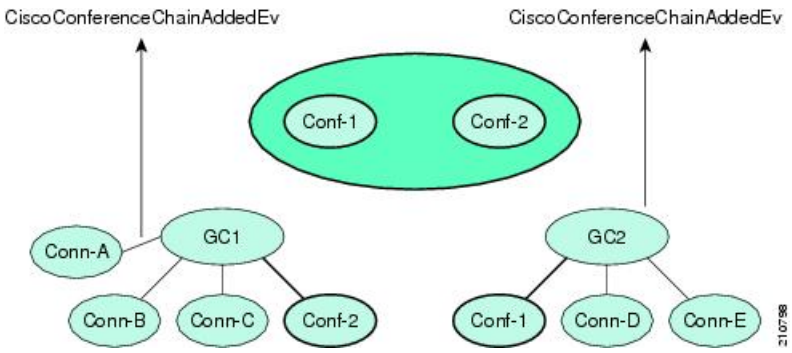
The following figure shows parties A, B, and C in conference call GC1 and parties C, D, and E in conference call GC2.

Figure 3: Calls Prior to Chaining



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

Figure 4: 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 (for the getConferenceChain() interface)
- CiscoConferenceChain
- CiscoConferenceChainAddedEv
- CiscoConferenceChainRemovedEv

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, CiscoProvTerminalRemoteDestinationChangedEv, CiscoProvider, CiscoTerminalProtocol.

Message Sequences

See CTI Remote Device.

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
- CiscoAnnouncementStartedEv
- CiscoAnnouncementEndedEv
- CiscoAnnouncementErrorEv
- CiscoFeatureReason

Message Sequences

See Play Announcement.

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

Message Sequences

Verify Remote Destination Support

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)

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

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

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
- CiscoMultiMediaCapabilityInfo
- CiscoProvTerminalMultiMediaCapabilityChangedEv
- CiscoTerminal

Message Sequences

See CTI Video Support.

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 124.

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

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	

Transfer on Same Lines	Transfer Across Lines
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

Message Sequences

See Direct Transfer Across Lines Use Cases

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
- CiscoTermDNDStatusChangedEv
- CiscoTermEvFilter
- CiscoCall
- CiscoConnection
- CiscoRouteSession
- CiscoTermInServiceEv

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; CiscoCall; CiscoTermEvFilter

Message Sequences

DND-R

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 and CiscoConnection

Message Sequences

See Drop Any Party Use Cases

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 To view the message flow for Dynamic CTIPort Registration Per Call, see Message Sequence Charts

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, CiscoConnection

Message Sequence

E911 Teleworker

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.

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, CiscoConnectionUniqueIDChangedEv.

Message Sequences

End to End Call Tracing

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

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, CiscoTerminal

Message Sequences

Extension Mobility Cross Cluster

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

Message Sequences

Extension Mobility Login Username

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, CiscoConnection, CiscoAddress

Message Sequences

External Call Control

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.

Message Sequences

End to End Session ID for Calls

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're 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. You can only witness this loss during the certificate downloads and opening of a Cisco Unified Communications Manager JTAPI provider. FIPS does not affect anything when 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 Unified CM server (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 Unified CM server or cluster secure or FIPS-compliant.

Likewise, having the Unified CM 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 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.

Note From Release **14SU2** to **15SU1**, **the BCFIPS libraries are**:

- bc-fips.jar (version 1.0.2.3)
- bctls-fips.jar (version 1.0.12.3)
- bcpkix-fips.jar (version 1.0.5)

Note From Release **15SU2** onwards, **the BCFIPS libraries are**:

- bc-fips.jar (version 2.0.0)
- bctls-fips.jar (version 2.0.19)
- bcpkix-fips.jar (version 2.0.6)
- bcutil-fips.jar (version 2.0.1)

If you're upgrading the BCFIPS jars in your runtime environment to the latest supported (2.x as aforementioned) and an application is using an older jtapi.jar (earlier than 15SU2), while setting up the application environment, you must explicitly set "org.bouncycastle.jsse.fips.allowRSAKeyExchange" to True as a runtime system property to ensure that CAPF operation succeeds.

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

If your application contains a lib folder where third-party libraries are stored, the classpath looks as follows:

- For the JTAPI plugin using RSA libraries (refer above for library usage information 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 (refer above for library usage information 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 (refer above for library usage information 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 that are 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, CiscoProvider, and CiscoJtapiProperties

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 Unified CM 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.

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, CiscoConnection, CiscoAddress, CiscoJtapiProperties

Message Sequences

Hunt List

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

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

CiscoTerminal

CiscoTermEvFilter

Message Sequences

Hunt Log Status for Phone Devices

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

- CiscoAddrIntercomInfoRestorationFailedEv
- Related Documentation
- CiscoCall
- CiscoProvider
- CiscoTermEvFilter
- CiscoTerminal
- CiscoTerminalConnection
- CiscoTermDeviceStateWhisperEv

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 a nonsecure JTAPI connection in IPv6 only deployments. 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 supports the following functions:

- Cisco Unified JTAPI exposes the 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 a 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.

Note We do not support a secure JTAPI connection in IPv6 only deployments with TLS.

Interface Changes

See CiscoTerminal

Message Sequences

See IPv6 Support and IPv6 Support

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

Message Sequences

iSac Codec

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.

Message Sequences

See CiscoJtapiProperties.

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 88.

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 84 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	
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.conference(GC2)	A calls B1[GC1 = GolbalCallID1] GC1: Connection A1 Conn1 GC1: Connection B1 Conn2 B2 calls C[G = GolbalCallID2] GG2: Connection B2 Conn3 GC2: Connection C Conn4 GC1.conference(GC2)

Join on Same Lines	Join Across Lines
Events Delivered to Application (assuming all parties are observed)	
GC1: CiscoConferenceStartEv [getConferenceControllerAddress() returns B1] ConnCreatedEv for C ConnConnectedEv for C CallCtlConnEstablishedEv for C TermConnCreatedEv for T3(Address C) CiscoConferenceEndEv GC2: CiscoConferenceStartEv [getConferenceControllerAddress() returns B1] TermConnDroppedEv for T2(Address B1) CallCtlTermConnDroppedEv for T2(Addresss B1) ConnDisconnectedEv for B1 CallCtlConnDisconnectedEv for B1 TermConnDroppedEv for T3(Address C) ConnDisconnectedEv for C CallCtlConnDisconnectedEv for C CallCtlTermConnDroppedEv for T3(Address C) CiscoConferenceEndEv CallInvalidEv CallObservationEndedEv **Note** GC2 - Disconnect events are for Address B1 on Terminal T2	GC1: CiscoConferenceStartEv [getConferenceControllerAddress() returns B1] ConnCreatedEv for C ConnConnectedEv for C CallCtlConnEstablishedEv for C TermConnCreatedEv for T3(Address C) CiscoConferenceEndEv GC2: CiscoConferenceStartEv [getConferenceControllerAddress() returns B1] TermConnDroppedEv for T2(Address B2) CallCtlTermConnDroppedEv for T2(Addresss B2) ConnDisconnectedEv for B2 CallCtlConnDisconnectedEv for B2 TermConnDroppedEv for T3(Address C) ConnDisconnectedEv for C CallCtlConnDisconnectedEv for C CallCtlTermConnDroppedEv for T3(Address C) CiscoConferenceEndEv CallInvalidEv CallObservationEndedEv **Note** GC2 - Disconnect events are for Address B2 on Terminal T2
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

Message Sequences

See Connected Conference or Join Across Lines Use Cases - New Phones Behavior

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

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

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

Message Sequences

See Logical Partitioning Feature Use Cases

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 To view the message flow for media termination at route point, see Message Sequence Charts

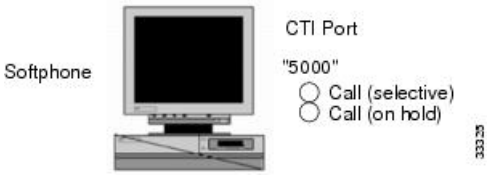
Media Termination Extensions

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 5: 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

Message Sequences

See Enhanced MWI Use Cases

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 To view the message flow for Modifying Calling Number, see Message Sequence Charts.

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

Message Sequences

See Native Queuing.

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:

Procedure

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 contains an identifier for the requestor party.
- The method getConferenceControllerAddress returns the terminal connection of the requestor.
- The new method getOriginalConferenceControllerAddress() for CiscoConferenceStartEv 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

Message Sequences

See Park Monitoring Support

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

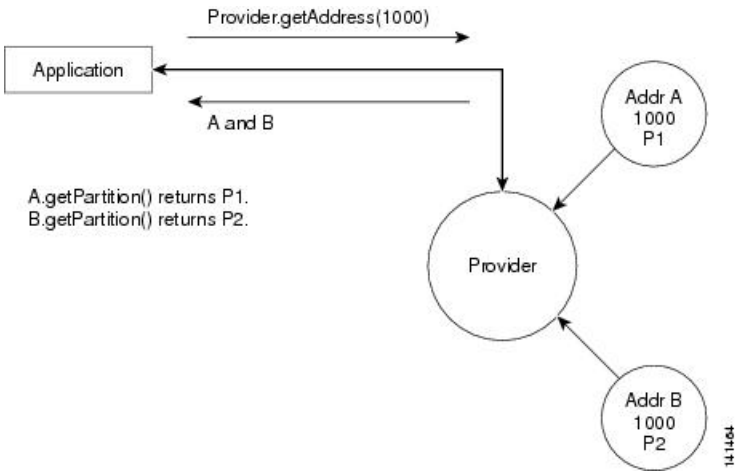
A new method is provided in this class with the following signature.

```
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 6: 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.

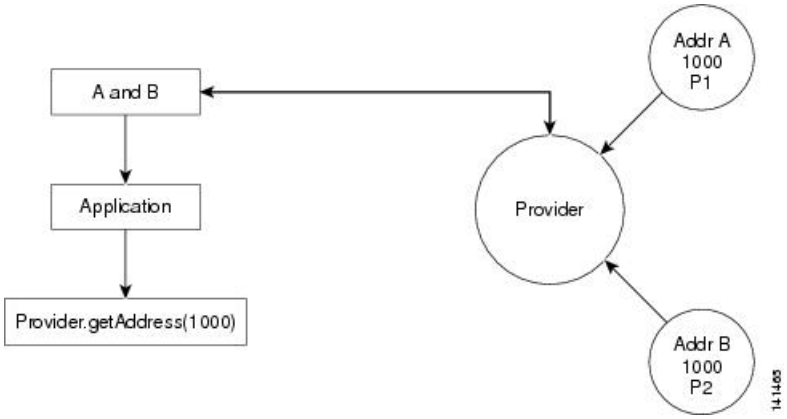
CiscoProvider Interface

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.

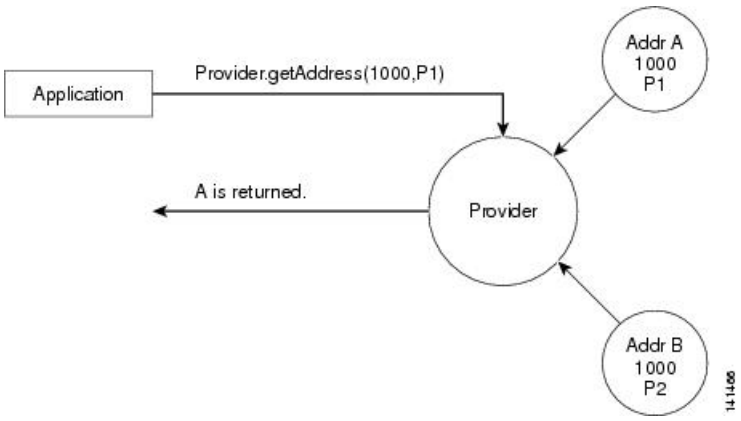
Figure 7: provider.GetAddress() Returns Two Address Objects



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 8: 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 To view the message sequences for partitions support, see Message Sequence Charts

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 28; CiscoJtapiExceptions, on page 27

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, CiscoTone

Message Sequences

See Play Zip Tone

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.

Procedure

Step 1 If you are running Windows 2000, follow the steps in QoS Setup on Windows 2000, on page 113.

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 113.

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 To view the message flow for QoS, see Message Sequence Charts.

QoS Setup on Windows 2000

If you are running Windows 2000, follow these steps.

Procedure

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.

Procedure

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
- Related Documentation
- CiscoCall
- CiscoProvider
- CiscoProviderCapabilities
- CiscoProviderCapabilityChangedEv
- CiscoProviderObserver
- CiscoRecorderInfo
- CiscoTerminalConnection
- CiscoTermConnRecordingTargetInfoEv

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 To view the message flow for Redirect Set Original Called ID, see Message Sequence Charts

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

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.

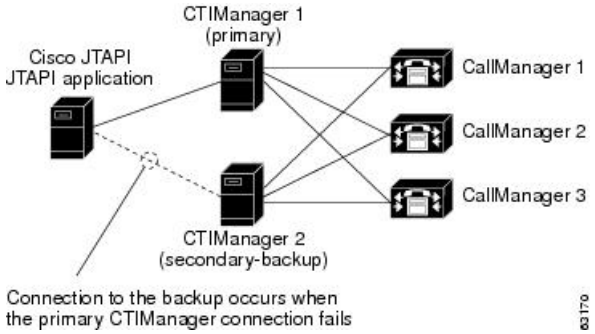
Redundancy in CTI Managers

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 9: 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.

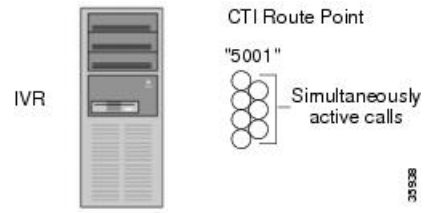
JTAPI implementation of adjunct Routing, as described in the call center package, includes the following actions:

- 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 10: 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.

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, CiscoTermConnMonitorInitiatorInfoEv, CiscoTermConnMonitorTargetInfoEv, CiscoAddrMonitorTerminatedEv, CiscoAddrEvFilter,

Message Sequences

Secured Monitoring Use Cases, Secured Recording

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 To view the message flow for SelectRoute Interface Enhancement, see Message Sequence Charts.

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

Message Sequences

selectRoute() with Calling Search Space and Feature Priority

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 To view the message flow for shared lines, see Message Sequence Charts

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
- Related Documentation
- CiscoCall
- CiscoMediaTerminal
- CiscoMonitorTargetInfo
- CiscoMonitorInitiatorInfo
- CiscoProvider
- CiscoProviderCapabilities
- CiscoProviderCapabilityChangedEv
- CiscoRecorderInfo
- CiscoTerminalConnection
- CiscoTermConnMonitorInitiatorInfoEv
- CiscoTermConnMonitorTargetInfoEv

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, CiscoTermConnMonitorInitiatorInfoEv, CiscoTermConnMonitorTargetInfoEv, CiscoAddrMonitorTerminatedEv, CiscoAddrEvFilter

Message Sequences

Secured Monitoring Use Cases, Secured Recording

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

Message Sequences

See Single Sign-On

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

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, CiscoMediaOpenIPPortEv, CiscoMediaOpenLogicalChannelEv, CiscoJtapiException

Message Sequences

See SIP Trunk Early Offer

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

Message Sequences

See Support for Cisco Unified IP Phone 6901

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 and CiscoProviderCapabilityChangedEv

Message Sequences

See Support for Cisco Unified IP Phone 6900 Series

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

Message Sequences

See Swap or Cancel and Transfer or Conference Behavior Change

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, CiscoAddrEvFilter, CiscoAddrVoiceMailPilotChangedEv, CiscoTerminal, CiscoProvFeatureID, CiscoProvTerminalRegisteredEv, and CiscoProvTerminalUnRegisteredEv.

Message Sequences

See Terminal and Address Capability Settings Use Cases

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

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

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 the provider continue to exist, after a certain number of retries, the 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.

Note From Release 15SU2 onwards, JTAPI supports TLS 1.3. However, TLS 1.3 support depends on the Java version used. Java version 1.8 (minor version 261) onwards supports TLS 1.3. CiscoJTAPI supports Java 1.8.

The supported ciphers when communicating with CAPF are as follows:

FIPS enabled:

```
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
```

FIPS disabled:

```
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
```

The supported ciphers when communicating with CTI Manager are as follows:

```
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA
```

Note From Release 15SU3 onwards, CiscoJtapiProperties interface has been enhanced to update the minimum TLS version that an application uses while establishing a secure communication channel to CTI Manager and CAPF.

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 interface 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 a 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 the 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

KEY	VALUE
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 server 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.

- 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 the 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.

```
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

Message Sequences

See Unrestricted Unified CM

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.

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.

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.

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.
- 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.
- 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.
- 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.
- 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
- CiscoMasterKeyIndicator
- CiscoMultiMediaCapabilityInfo
- CiscoMultiMediaConnectionMode
- CiscoMultiMediaEncryptionKeyInfo
- CiscoMultiMediaProperties
- CiscoMultiMediaStreamsInfoEv
- CiscoMultiMediaType
- CiscoProvTerminalMultiMediaCapabilityChangedEv
- CiscoRTPPayload
- CiscoRTPProperties
- CiscoTermEvFilter
- CiscoTerminal

Message Sequences

See Video Capabilities and Multi-Media Information.

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.