Caveats

This appendix provides details of the JTAPI caveats that are common across releases and those that are release-specific.

Caveats for All Releases

This section lists the caveats that are common for all JTAPI releases and contains these topics:

• Unsupported CTI Events for SIP Phones, on page 6

Single Versus Multiple CallObserver Clarification

There are two primary ways to observe addresses with Cisco JTAPI's CallObservers: an application can observe all of the addresses with a single CallObserver object or it can have a separate CallObserver object for each of address. These two approaches cause slightly different events to be seen, mostly with regard to reason codes.

When an application uses a single CallObserver to observe all the addresses, they are connected with one object. When A calls B, both the events at A and at B are sent to the same CallObserver object. If B redirects to C, and C was observed with the same object, all its events are delivered to the same observer. The application observes the CallCtlConnOfferedEv to C with reason REDIRECT, because the observer at C knew all about the previous events on the call.

Conversely, when an application uses an independent CallObserver for each Address, this information is not so easily shared. When A calls B, call events of A go to the observer for A, and B's go to the observer for B. They each know about the other end, for example A will know that B is ringing, but they are no longer the same observer. When B redirects the call to C, the observer at C knows absolutely nothing about the call. The observer at C was not involved in the original call at all, and does not know who is on it, what events had happened previously. This information has to be made up by JTAPI to build an accurate call model at C. All the call events for a basic call between A and B have to be simulated so that the call model, from C's perspective makes sense.

This is done by using a snapshot event. JTAPI looks at the call, in this case the one between A and B, and figures out what events have to have happened for the call to exist the way it does. This makes up up the basic call events required, and give them to the observer on C, so that it can build a proper call model.

Since this event set is made up by JTAPI, the reason codes are not available. For example, if A had originally called D, and D redirected to B, the made up snapshot event set would not be concerned with the redirect at all. JTAPI does not store this information anywhere, and when it generates a snapshot, it creates the simplest event set possible to recreate the call model, and reports all the events with reason NORMAL.

So, when A calls B, and then B redirects to C, the observer on C gets a snapshot event that allows it to recreate the call model for a basic call from A to B. Also in this snapshot event is the CallCtlConnOfferedEv for C. As part of the snapshot, this event comes in with reason NORMAL, even though it is the result of a redirect. CallObservers on A or B will see the CallCtlConnOfferedEv for C with reason REDIRECT, but there is no way for the observer at C to know that.

This creates a noticeable difference in the reason codes available to applications depending on how they implement their CallObservers. There have not been any issues regarding this from the customer side.

This is the way it has been since Cisco JTAPI's inception and this is a clarification of the existing behavior.

SIP and SCCP Dialing Differences with Overlapping Directory Number Patterns

An overlapping Directory Number Pattern is when one Directory Number is a part of a longer Directory Number. For example, a Directory Number 1000 overlaps a Directory Number 10001. Cisco Unified Communications Manager (Cisco Unified CM) and JTAPI both support overlapping Directory Number patterns, but there are some important things to note regarding this.

When you dial 1000 from a normal phone, there is a delay. The Cisco Unified CM does not know if you want to dial 1000 or 10001, so it waits for you to make a choice. This Digit Analyzer waits for 15 seconds if you press nothing else. During this time, nothing happens, no call is extended, and it just sounds like a dead call

for the calling party. This can be short circuited by hitting # to let Cisco Unified CM know that you actually intend to call 1000. The 15 second wait is a configurable service parameter, known as T302 Timer, and can be set as low as 3 seconds.

JTAPI using SCCP phones avoid the Digit Analyzer entirely by communicating directly with Cisco Unified CM. JTAPI sends the intended Directory Number when it is making a call, and if it sends 1000, it means only that, and Cisco Unified CM knows it will not be dialing any more digits.

JTAPI using SIP phones is quite different. JTAPI communicates with the phone, which then communicates with the Cisco Unified CM. The phone takes care of the dialing, and JTAPI will pass it the digits 1000 to dial. Due to this, JTAPI cannot avoid the Digit Analyzer, and is subject to the T302 wait outlined above. JTAPI sits idly and waits while the Digit Analyzer figures out that the SIP phone actually wants to dial 1000.

Apart from this, by default the JTAPI CTI Postcondition value is also set to 15 seconds. This means that when JTAPI sends a request to the CTI layer, it waits for 15 seconds before it assumes something has gone wrong, and throws a timeout exception. This means that the delay for Digit Analysis for overlapping DN patterns is very likely to cause JTAPI to time out.

The Digit Analysis delay cannot be completely removed for SIP phones, but this problem can be greatly mitigated through the use of service or jtapi.ini parameters. As noted above, the T302 Timer for Digit Analysis can be set as low as three seconds, which is much lower than the 15 it takes JTAPI to time out. You can also increase the JTAPI CTI Postcondition timeout to 20 seconds in the jtapi.ini file. This issue can also be avoided by not having overlapping DNs.

Translation Pattern Support

If the callingparty transformation mask is configured for a Translation Pattern that is applied to the controlled addresses of JTAPI application, the application may observe some extra connections being created and disconnected when the application observes both calling and called party. Otherwise, a connection is created for the transformed callingparty and CiscoCall.getCurrentCallingParty() returns the transformed calling party address when the application observes only the called party. In general, JTAPI has a problem in creating an appropriate call connection and may not be able to provide correct callinfo such as currentCalling, currentCalled, calling, called, and lastRedirecting parties. For example, Translation Pattern X is configured with calling party transformation mask Y and calledparty transformation mask B; A calls X, and call goes to B. In this scenario:

- If the application observes only B, JTAPI creates connection for Y and B, and CiscoCall.getCurrentCallingParty() returns Address Y.
- If the application observes both A and B, connections for A and B are created, connection for Y is temporarily created and dropped, and CiscoCall.getCurrentCallingParty() returns Address Y.

Other inconsistencies could occur in callinfo if more features are used for basic call. It is recommended that you do not configure callingparty transformation mask for Translation Pattern which might get applied to JTAPI Application controlled Addresses.

DT24+ Limitation with PRI NI2 Trunk

When a PRI NI2 trunk used by DT24+ gateway is involved in a call scenario between two clusters, for example, A from cluster-1 calls B in cluster-2 via DT24+ PRI NI2 trunk, the LastRedirectAddress and CalledAddress may not be accurate on B’s side. Besides, if there are any changes for A’s side of the call in cluster-1 due to redirect, transfer, or forward, the changed information is not propagated to B’s side due to protocol limitation of PRI NI2 truck.

Connection for Park Number Not Created

JTAPI does not create a Queued state connection for Park Number if the call is parked across the gateway. There are two possibilities here:

1. If CLI is configured, application sees an Unknown connection
2. If CLI is not configured, the calling does not see any Unknown connection.

For Example, If A calls B across gateway (with CLI configured) and B parks the call then A sees an unknown connection instead of a connection (with STATE = QUEUED) for Park Number.

But, if A calls B across gateway (with no CLI configured) and B parks the call then A does not see any new connection.

Inconsistency Between SIP and SCCP Phone

sendData() API on CiscoTerminal is used to send data to the phone. In case of SIP phones, if invalid byte data is sent by the application, the method throws PlatformException. However, in case of SCCP phones, the byte data sent in the request is not validated, and would return successfully without throwing an exception.

Failure to Route Calls Across Destinations

When a call is redirected to a device outside the cluster over an H323 gateway that is out of service, before call control can determine the Out-of-Service status of H323 gateway, the call is disconnected. This is because the default value of the service parameter CTI NewCallAccept timer is four seconds where as call control takes five seconds to determine that the gateway is out of service, so calls are disconnected due to expiration of CTI NewCallAccept timeout.

Implications of the above behavior is seen in JTAPI selectRoute() API which internally uses CTI redirect API to route the call. If applications specify multiple destinations with selectRoute() and the first destination is across an out-of-service H323 gateway, the call fails before JTAPI can route the call to the second destination. Hence JTAPI, cannot route the call to the second route specified in selectRoute() interface call.

To avoid this, the value of CTI New Call Accept Timer service parameter can be set as greater than H225 TCP Timer service parameter.

Incorrect Return Value for getCallingAddress()

In a transfer scenario, where caller and transferController are not observed, that is, where A calls B and transfers the call to C and the application is observing only C then before the transfer, JTAPI will not have any information about the first call (that is, call from A to B). So, when the transfer feature is invoked, the calling and called address are B and C respectively. On completing the transfer, application updates callInfo and JTAPI exposes the correct parties through getCurrentCallingAddress(), getCurrentCalledAddress(), getModifiedCallingAddress(), and getModifiedCalledAddress(). However, getCallingAddress() API which should return the original calling address still reports B, that is, the original calling party of B to C call.

To avoid this issue, application can observe the controller as well, so that JTAPI expose the correct party (that is A, in this case) with getCallingAddress() API.

Call Fails to Disconnect Held Shared Line

In a scenario where A calls B (B is a shared line present on terminals T1 and T2); privacy is set as ON for T1 and initially CUCM service parameter Enforce Privacy Settings on held calls is set to True. B(T1) answers and goes to Talking state while B(T2) goes to Passive state (TermConn = Passive; CallCtlTermConn = InUse). B(T1) puts the call on hold, since Enforce Privacy Setting on held calls is set to True, B(T2) remains in passive state (TermConn = Passive; CallCtlTermConn = InUse). Now the service parameter Enforce Privacy Settings on held calls is set to False. This does not trigger any change in the state of TerminalConnection, so B(T2) still remains Passive-InUse (TermConn = Passive; CallCtlTermConn = InUse). At this point, if the application sets the requestController as B(T1) and disconnects the call at B, the connection of B is not disconnected and call does not go IDLE. Even on the phones, the call on A remains in Established state while the other party in call is B(T2) which remains in Passive-InUse (TermConn = Passive; CallCtlTermConn = InUse) state. Call is cleared when A disconnects the call.

Limitation with sendData() API on CiscoTerminal

If JTAPI applications make simultaneous back to back requests for sendData() API on the same CiscoTerminal, without any delay between requests, then some of these requests may fail. Applications cannot determine whether a request was successful or not, as Cisco JTAPI API returns successfully as soon as the phone receives data and does not wait for a response from the phone. Also, the IP phone might display a blank screen on sending simultaneous requests to send data.

To avoid these issues, JTAPI applications should ensure some time delay between two successive sendData() requests while pushing XSI data to the IP phones via Cisco JTAPI.

Limitation in Using ; (Semi-Colon) and = (Equal) in User ID and Password

Sun JTAPI 1.2 specification does not support use of the semicolon ';' and equals ' = ' characters when populating the Host Name, UserID, and Password fields in string used as parameter in getProvider() method. If ';' or ' = ' are used in these fields, items such as 'pass = word' or 'pass;word' are treated as 'pass' and your request could fail and you must not use these characters in userid and password fields.

Connection to Unknown Address When Unparking a Conference Call

When a conference call is parked, JTAPI call will have connection to the remaining parties in the call. When this call is unparked using the UnPark API or connect API, connections to unknown address will be temporarily created. This connection to unknown address will be disconnected when un park is completed.

The following will be seen in JTAPI traces:

2002 : (P1-InProv) 103023/1 ConnCreatedEv Unknown:null:4 187 Cause:100 CallCtlCause:0 CiscoCause:31 FeatReason:10 CAUSE_NORMAL

....

2002 : (P1-InProv) 103023/1 ConnDisconnectedEv Unknown:null:4 187 Cause:100 CallCtlCause:0 CiscoCause:31 FeatReason:10 CAUSE_NORMAL

CTI Redirect to Voice Mail Wont Work with QSIG

When applications redirect a call to voice mail through a QSIG trunk, voice mail will not play the voice prompt of the redirecting party. A generic prompt asking the user to enter the voice mail box is played. This is due to the fact that CTI redirect doesn't pass correct original called party and redirecting party to the voice mail system. Applications can work around this by using a SIP trunk.

Phone A calls Phone B (CUCILYNC in Deskphone mode). Toast popup of "envelope" appears for incoming call.

User clicks on the envelope, redirecting the call across an H.323 trunk with QSIG tunneling. Unity/Unity Connection won't recognize original called party, so call will not go to user's voice mailbox. This is because Re-direct/Original Called Party information not carried across the trunk when CTI redirect is used.

CiscoAddress.getForwarding() Returns Correct Value Only for In-Service Addresses

Although the Sun JTAPI 1.2 specification does not specify any preconditions for CallControlAddress.getForwarding(), the implementation of this method in Cisco JTAPI will return a correct value only if the address is in service. When invoked on an out of service address this method will just return a NULL value.

Unsupported CTI Events for SIP Phones

The following CTI events are not generated for SIP phones. Third party applications that expect these call events should use SCCP phones:

- CallOpenLogicalChannelEvent
- CallRingEvent
- DeviceLampModeChangedEvent
- DeviceModeChangedEvent
- DeviceDisplayChangedEvent
- DeviceFeatureButtonPressedEvent
- DeviceKeyPressedEvent
- DeviceLampModeChangedEvent
- DeviceRingModeChangedEvent

Caveats for Release 9.1(1)

This section lists the JTAPI caveats for Release 9.1(1).

- Connection for Park DN While UnPark, on page 7

Connection for Park DN While UnPark

Cisco JTAPI will no longer create the temporary connections for the Park/DPark number in the final call when a parked call is unparked.

This change in behavior will be seen in and above the following Cisco JTAPI versions - 8.5.1.10000-10, 8.6.1.10000-4, 8.6.2.10000-2 and 8.6.2.98000-2

Scenario:

GC1: A calls B, B parks a call is parked at 1000

GC2: B unparks

“Preserve globalCallId for Parked Calls” is set to false

Previous behavior on unpark:

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 ConnCreatedEv 1000
GC2 CallCtlConnEstablishedEv B
GC2 ConnCreatedEv A
GC2 ConnDisconnetedEv 1000
GC2 CallCtlConnEstablishedEv A
GC1 CallChangedEv
GC1 ConnDisconenctedEv 1000
GC1 ConnDisconenctedEv A
GC1 CallInvalidEv
```

Current behavior on unpark: Connection for 1000 will not be created in GC2

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 CallCtlConnEstablishedEv B
GC2 ConnCreatedEv A
GC2 CallCtlConnEstablishedEv A
GC1 CallChangedEv
GC1 ConnDisconenctedEv 1000
GC1 ConnDisconenctedEv A
GC1 CallInvalidEv
```

"Preserve globalCallId for Parked Calls" is set to true

Previous behavior on unpark:

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 ConnCreatedEv 1000
GC2 CallCtlConnEstablishedEv B
Gc2 CallChangedEv
GC2 ConnDisconnetedEv 1000
GC2 CallCtlConnDisconnectedEv 1000
GC2 ConnDisconnetedEv B
GC2 CallCtlConnDisconnectedEv B
GC2 CallInvalidEv
GC1 ConnCreatedEv 1000
GC1 ConnCreatedEv B
Gc2 ConnDisconnectedEv 1000
GC1 CallCtlConnEstablishedEv B
```

Current behavior on unpark: Connection for 1000 will not be created in GC1

```
GC2 CallActiveEvGC2 ConnCreatedEv B
GC2 CallCtlConnEstablishedEv B
GC1 ConnDisconnectedEv 1000
GC1 CallCtlConnDisconnectedEv 1000
Gc2 CallChangedEv
GC1 ConnCreatedEv B
GC1 ConnConnectedEv B
GC1 CallCtlCallEstablishedEv B
GC2 ConnDisconnetedEv B
GC2 CallCtlConnDisconnectedEv B
GC2 CallInvalidEv
```

Caveats for Release 8.6(1)

This section lists the JTAPI caveats for Release 8.6(1).

• Limitation While Using a Cisco Telepresense MCU, on page 8

Limitation While Using a Cisco Telepresense MCU

In scenarios, where a conference chaining is done across cluster, the number of connection seen by the application might not be correct and application may not see the connection for the external conference bridge getting created.

Example:

```
A,  B and B' are in Cluster 1D is in cluster 2
Application is observing D
GC1: A calls D; Call offered on D and D answers
GC2: D consults B; B answers
B' CBarges into the call
D completes conference - GC1.conference(GC2)
```

After the conference is complete the application will see only the connection for A and D but not that of the conference bridge.

Caveats for Release 8.5(1)

This section lists the JTAPI caveats for Release 8.5(1).

• Discouraged Use of JTAPIProperties.updateCertificate(), on page 8
• Delete SecurityProperties Before Re-Use, on page 9
• No ConnDisconnectedEv Event When Call Is Rejected, on page 9

Discouraged Use of JTAPIProperties.updateCertificate()

Applications are encouraged to move away from using the JTAPIProperties.updateCertificate() method to download certificates. The JTAPIProperties.setSecurityPropertyForInstance() method is superior for most applications, as it will store the security information in the jtapi.ini file, giving all the information to JTAPI

whenever the application chooses to connect securely later. This information is critical for JTAPI to use the correct settings to create a secure connection, and if an application chooses to use the JTAPIProperties.updateCertificate() method, then the information will not be stored in the jtapi.ini file.

Applications that have no reason not to, should move to setSecurityPropertyForInstance().

If an application is intentionally using the updateCertificate() method to avoid the use of jtapi.ini configurations, then the application must provide the information in the JTAPI Provider String, when it is first initializing JTAPI.

The required fields are:

- InstanceID
- CertStorePassphrase
- CAPF
- CAPFPort
- TFTP
- TFTPPort
- CertPath

If an application chooses to use the updateCertificate() method and chooses not to provide the required security information in the Provider String, the behavior of JTAPI is not guaranteed.

Delete SecurityProperties Before Re-Use

The JTAPIProperties.setSecurityPropertyForInstance() method downloads certificates to disk, and stores security information related to them in the jtapi.ini file. If an application is interested in downloading new certificates to disk, changing the security information for the certificates, or both, is it recommended that they invoke JTAPIProperties.deleteSecurityPropertyForInstance() before doing so. This method will delete the certificates from disk, and remove the related SecurityProperty from the jtapi.ini file.

This will ensure a fresh start for the new set of certificates, and help to eliminate any errors that could arise from "stale" certificates lingering around on disk.

No ConnDisconnectedEv Event When Call Is Rejected

For CUCM version 8.5.1 and later versions, when a call is made from A (observed) to B (unobserved) from application and if call is rejected for any reason, application will get the ConnFailedEv/CallCtlConnFailedEv events for the calling party, but there might be just one connection created in JTAPI for the calling party and there might not be any ConnDisconnectedEv/CallCtlConnDisconnectedEv events for called party. In addition, the ConnDisconnectedEv/CallCtlConnDisconnectedEv events for the calling party would be generated only after the calling party device/phone clears the call which itself can take 30 secs or more. Therefore, if application does not want to wait for the Disconnect events, upon getting the Failed events, it can simply clear the call directly after catching the failure from its call.connect() request as 'PlatformExceptionImpl caught: Could not meet post conditions of connect()'.

Caveats for Release 8.0(1)

This section lists the JTAPI caveats for Release 8.0(1).

Globalized Calling Party Number

This caveat is a further clarification for Globalized Calling Party Number. With 8.0(1), there have been changes to the Call Processing and CTI layers that reflect the globalized calling party values that you see on the station more accurately. There are still some serious limitations with globalized parties:

- The Globalized Calling Party Number is available on the called side.
- The Globalized Calling Party Number is determined only when the call is offered. This applies to basic calls and for calls involving features.
    - The Globalized Calling Party Number does not change, even if the calling party changes. This is a clarification for the existing caveat Globalized Calling Party Number.
- Globalized Calling Party will be applicable if the call is redirected, whether performed prior to call being connected or after call is connected.
- Globalized Calling Party will not be provided after the these features are completed: Transfer, Conference, Unpark, Auto Call Pickup.

Conference Interaction with Chaperone Results in Unsupported Conference Chaining

If both caller and Chaperone complete conference, where caller completes conference before chaperone then the scenario results in Conference Chaining. As of release 8.0(1), such scenarios are not supported by JTAPI and currently connections for all the parties along with ConferenceChain connection are shown in a single call.

For example, A calls B; call is intercepted by Chaperone (C1) which further consults B for conference. Before Chaperone completes conference, Caller(A) goes ahead and consults X for conference and completes the conference. After this, Chaperone(C1) completes the conference. This scenario would result in unexpected behavior from JTAPI Perspective. JTAPI could send CiscoConferenceStartedEv along with CiscoChainAddedEv. Also, after the conference, JTAPI shows connections for two Conference chains, A(caller), B, X, and C1(Chaperone) in the same call.

This will be fixed in future releases by having connections of caller and X in one call which is chained to a different call with connections for C1 and B. The fix requires changes in multiple components which are tracked with CDETS: CSCtc76213(CTI), CSCtc76222(TSP), CSCtc76223(Conference)

Wildcard Routepoint Interaction

Before 8.0(1), Wildcard Routepoint scenarios were not supported by JTAPI. This behavior is maintained in this release, in the spirit of Backward Compatibility, but is still unsupported. A new Service Parameter has been introduced in 8.0(1), WildCardDN as Called Party, which fixes several issues in the call model for Wildcard Routepoint scenarios. When this service parameter is turned on, Wildcard Routepoints are fully supported by JTAPI.

Inconsistent Address Type of ModifiedCalledAddress When a Call Is Made to a Hunt Pilot

When a call is made to a Hunt Pilot, although the API call.getCurrentCallingAddress() will return an address of type CiscoAddress.HUNT_PILOT, the address type of the address returned by the API call.getModifiedCalledAddress() will not be CiscoAddress.HUNT_PILOT.

This is because the modified address, as the name suggests, might be modified by application and getType() on modified address would return CiscoAddress.UNKNOWN or CiscoAddress.INTERNAL and JTAPI currently has no way to determine if that corresponds to a Hunt Pilot or not.

Caveats for Release 7.0.1

This section lists the JTAPI caveats for Release 7.0.1.

Inconsistency in getModifiedCallingAddress()

When a call is made to a shared DN (Address shared) on two Terminals (A and B), configured with different Calling Party Transformation CSS, and try to transform the Calling Party differently through two different Calling Party Transformation Patterns, then JTAPI does not provide modifiedCallingParty consistently. In such a scenario, both devices send localization signals to call control to transform the calling party number and the one picked by call control first is used to transform the same and JTAPI returns that through getModifiedCallingAddress().

For Example, +918055552222 (Globalized Calling Party) calls JTAPI observed shared Line 3100 on Terminals A and B. Device A is configured to transform the calling party by removing International escape character where as, B is configured to transform the calling party by removing International Escape Character and country code 91. Then JTAPI cannot guarantee whether getModifiedCallingAddress(), that is, the Localized Calling Party, will return 918055552222 or 8055552222.

Conference Behavior for Selected and Active Calls

Following is the behavior when application issues conference request but selected and active calls are not part of the conference request:

Active Call on a Terminal is always added to the resulting conference when conference is invoked on a call on any address on that terminal.

Consider B1 and B2 are addresses on same terminal

A --> B1- GC1

C --> B1- GC2

D --> B2- GC3 (active call)

The application invokes GC1.conference(GC2) and results in A-B1-C-D in conference with GC1, although call with D was not part of the conference request.

Active conference call on a terminal is added to the resulting conference when conference is invoked on a call on any line on that terminal. In this case, the active conference call becomes the surviving final call (provided the application specified primary call is not a conference call). In this scenario, the application specified primary call is cleared after the conference. It is possible that the application specified primary call may not join the resulting conference and in that case the call is not cleared after conference is over.

Consider B1 and B2 are addresses on the same terminal and conf1 is a conference call with A-B1-C in conference with B1 as the controller

B1 --> D - GC1 (on hold)

conf1 - GC2 (active call)

B2 --> E - GC3 (on hold)

The application invokes GC1.conference(GC2, GC3).

This results in A-B1-C-D-E in conference with GC2 as the surviving call. Although application had specified GC1 to be the primary call, GC1 does not survive after the conference.

The same behavior applies for user selected calls that are not part of the conference request, but become part of the resulting conference as mentioned above.

The same behavior would apply to a regular conference with common controller. Consider A, B, C, D as lines on different terminals

A-->B - GC1

C-->B - GC2

D-->B - GC3 (active call)

The application requests GC1.conference(GC2). This results in A-B-C-D in conference with GC1. Although direct call with D was not part of the conference request, D will join the conference

Change in GlobalizedCallingParty Behavior

getGlobalizedCallingParty() returns the correct globalized calling number only in case of a basic call. In a scenario where A calls B, B answers. A and B are connected. In this case, if application requests for getGlobalizedCallingParty(), the API returns the globalized number for A. In case of features such as transfer or redirect where any of the party gets updated, getGlobalizedCallingParty() may not return the correct globalized number.

Caveats for Release 6.0.1

This section lists the JTAPI caveats for Release 6.0.1:

Call History Might Get Lost When AAR Routes Over QSIG Trunk

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

Different Event Order If Consult Call Initiated on SIP Device

Scenario: Terminal A initiates a call to the shared line B/B' and which Initiates a consult call to Terminal C.

- If the Shared Line is SIP device then the call events are :
  - B (active) receives: **CallCtlTermConnHeldEv** > **CiscoTermConnSelectChangedEv** > **CallActive**
  - B' (remote-in-use) receives: **CiscoTermConnSelectChangedEv** > **CallActive** > **CallCtlTermConnHeldEv**
- If the Shared Line is SCCP device then the call events are:
  - **CiscoTermConnSelectChangedEv** > **CallCtlTermConnHeldEv** > **CallActive** on both the Terminals.

Caveats for Release 5.0

This section lists the JTAPI caveats for Release 5.0:

SRTP Support

The default sRTP policy used by IPPhones is different from the one published (standard) as part of libSRTP code. libSRTP code is a free download, available at http://srtp.sourceforge.net/download.html. If the correct srtp policy is not used, the end result is no audio at both ends.

The srtp_policy is used by media terminating endpoint to create a crypto context. It should match to encrypt and decrypt packets sent/received by IPPhones/CTIPorts. Phone is using one hardcoded srtp_policy for all phone types including sip phones.

```
policy->cipher_type  =  AES_128_ICM;policy->cipher_key_len  =  30;
policy->auth_type  =  HMAC_SHA1;
policy->auth_key_len  =  20;
policy->auth_tag_len  =  4 ; // changed to 4 from 10;
policy->sec_serv  =  sec_serv_conf_and_auth;
```

JTAPI clients doing their own media termination with SRTP, must use the above policy to create a crypto context. The default policy published as part of libSRTP (the standard policy documented as part of RFC) is not same as used by Cisco Unified IP Phone. Phones use the modified one to optimize the bandwidth. The sRTP policy must be part of negotiation between the endpoints but right now only one is supported and ccm does not support the negotiation, hence applications need to use the above policy to terminate media.

Partition Support

When same Directory Number with different partitions exist in a CallManager, getAddress(String number) API in JTAPI normally returns the first matching Address object which has the same Directory Number. This is done to maintain BWC in case of no partitions configured in the system.

TLS Security

When client certificate is installed, server certificates are also downloaded from CallManager TFTP server. However, server certificate is only verified for currect signature and server trust is not established. Hence it is recomented that first time download of certificate is done in trusted network.

CiscoFeatureReason

When a call redirectred across a GW is offered at an address, getCiscoFeatureReason() returns REASON_FORWARDNOANSWER.

Within the same cluster when a call is redirected, the redirect destination sees REASON_REDIRECT as the CiscoFeatureReason. However, when a call is redirected across a cluster, through a Q.931 trunk, at the redirect destination getCiscoFeatureReason() returns REASON_FORWARDNOANSWER(as per Q.931 standard).

Scenario: A, B and C are registered to 3 clusters. A calls B, B answers and redirects the call to C. When call is offered at C, getCiscoFeatureReason() will return REASON_FORWARDNOANSWER.

Unicode Issue in Calls Involving SIP Trunks

In SIP call scenarios, where the call comes back to the call manager from the SIP proxy over a SIP trunk, since the call manager is totally dependent on the SIP, messages to populate any names and since the SIP protocol has no way of populating both ASCII and Unicode, the passed-in name is just ASCII and the same

gets sent to JTAPI. Hence, in such call scenarios, the user will not be able to see Unicode names since this is not under the control of JTAPI.

As a result the following APIs on CiscoCall interface will return only ASCII values instead of Unicode in such scenarios.

```
public String getCurrentCalledPartyUnicodeDisplayName();public String
getCurrentCallingPartyUnicodeDisplayName();
```

For SIP phone specific issues please refer to differences in SIP support section.

Join Across Lines: Conference Two or More Addresses on Same Terminal

For Unified Communications Manager releases 6.x and 5.x, if applications try to conference two or more addresses on the same terminal, based on the order of participants in the request, application may receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT for the conference request and later the conference may be created successfully with some of the participants. In such a case, there is no guarantee which one of them joins the conference. But the conference is created with only one of the address on a terminal and others are ignored. This depends on how this feature request is processed in different CUCM releases.

Below are the details of two scenarios affected:

Note This issue has been resolved in 7.x using CSCsj06488 and CSCsj06533.

1. Consider B1 and B2 are different address on the same terminal TB.

   A ->B1 - GC1

   A ->B2 - GC2

   A ->C - GC3

   Application issues a conference request GC1.conference(GC2, GC3)

   In 5.x and 6.x, application will receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT, however A, B1, C come into conference, and the normal set of events delivered in a conference scenario are seen like mentioned below:

   GC1 CiscoConferenceStartEv

   GC2 TermConnDroppedEv TB

   GC2 CallCtlTermConnDroppedEv TB

   GC2 ConnDisconnectedEv B1

   GC2 CallCtlConnDisconnectedEv B1

   GC1 CallCtlTermConnTalkingEv TB

   GC2 CiscoCallChangedEv

   GC1 ConnCreatedEv C

   GC1 ConnConnectedEv C

GC1 CallCtlConnEstablishedEv C

GC1 TermConnCreatedEv TC

GC1 TermConnActiveEv TC

GC1 CallCtlTermConnTalkingEv TC

GC2 TermConnDroppedEv TC

GC2 CallCtlTermConnDroppedEv TC

GC2 ConnDisconnectedEv C

GC2 CallCtlConnDisconnectedEv C

GC2 CallInvalidEv

GC1 CiscoConferenceEndEv

2. Consider B1 and B2 are different address on the same terminal TB.

   A ->B1 - GC1

   A ->B2 - GC2

   A ->C - GC3

   Application issues a conference request GC3.conference(GC1, GC2)

   In 5.x, Application will not receive any exception and the request would be procressed successfully. A, C, B1 would be in conference along with the regular set of conference events.

   In 6.x, Application will receive CiscoJtapiException.CONFERENCE_INVALID_PARTICIPANT, however A, C, B1 come into conference, and the normal set of events delivered in a conference

JTAPI Exposes Incorrect Information with getCallingAddress() and getCalledAddress()

In some scenarios where feature works on multiple calls (pickup, transfer, conference and so on) depending on Event Order or the parties observed, JTAPI may end up reporting incorrect call information to the applications. These scenarios are ones where surviving call is initially not there in JTAPI's provider domain or in scenarios where SurvivingCall goes invalid and is recreated in the middle of the feature operation. For such survivingCalls, JTAPI does not report correct information with getCallingAddress() and getCalledAddress(). Some of these scenarios are:

1. Transfer - A calls B; B transfers the call to C and application is observing only C
2. HuntList Transfer - In 8.x release if transfer is done, then surviving call can go invalid and recreated if caller is not observed.
3. PickUp scenario where survivingCall goes invalid and is recreated in middle of the feature
4. UnPark scenarios where caller is not observed and service parameter, Preserve globalCallId for Parked Calls, is set to true.

In general, this issue can happen with scenario where survivingCall is not in provider's domain initially or if is there and goes Invalid(CallInvalidEv is sent) and is recreated(CallActiveEv is sent) during the feature operation.

Caveats for Release 4.1

This section lists the JTAPI caveats for Release 4.1:

FAC-CMC

1. Forwarding should not be configured to a DN that requires FAC-CMC code. Forwarding requests will be successful, but calls will not be forwarded to these DNs and will be rejected.
2. Application should always terminate the code with #, otherwise system waits for T302 timer before extending the call. For these cases, application could get postConditionTimeOut exception for call.connect() or call.consult() but call may actually be offered. If apps need to avoid this, either all the digits with # terminated string are entered with post condition timeout (which is by default 15 sec in JTAPI Prefs UI) in the PlatformException or increase the postcondition timeout.
3. Two identical CiscoToneChangedEvents are sent to applications and second one needs to be ignored if both the codes are entered with # separated upon receiving the first event.

setConferenceController

The party that starts a conference by adding a new party acts as the original conference controller. Only the original conference controller can add new parties into the conference. If the original conference controller drops out of the conference, no other party in that particular conference call can add a new party. Although the conference controller cannot be changed while a conference call is going on, applications can determine which TerminalConnection acts as the conference controller when initially setting up a conference call via the CallControlCall.setConferenceController() method. The CallControlCall.getConferenceController() method returns the current conference controller, or null if there is none. If no conference controller is set, the implementation chooses a suitable TerminalConnection when the conferencing feature is invoked."

Consider the following scenario as an example:

A, B, C, and D belong to a conference call and all are in the TALKING state. A acts as the conference controller. A attempts to use the SetConferenceController API to change the conference controller to B, gets no error, and drops out of the conference. B then tries to add a new party, E, into the conference but cannot do so.

conference(Call[])

Applications can control which TerminalConnection acts as the conference controller when setting up a conference call via the CallControlCall.setConferenceController() method. The CallControlCall.getConferenceController() method returns the current conference controller, or null if there is none. If no conference controller is set initially, the implementation chooses a suitable TerminalConnection

when the conferencing feature is invoked. Only the original conference controller can add new parties to a conference call. Attempting to change the conference controller while a conference is going on will not take effect; however, no error gets thrown in the setConferenceController API

Interval During DTMF Digits

As per fix for CSCef05359, Change PlayDTMF to allow applications specify the time delay, now applications can configure the time delay during DTMF digits through Admin page, Service pameter, generate DTMF delay, for call Manager.

Shared Lines Support

Cisco JTAPI does not support configuration of same Directory Number from different partitions on the same or any device but configuration of different Directory Number from different partitions on the same device as well as different devices is supported.

CP Requires Previous Calls on the Device to Be in Connected Call State

CP requires previous calls on the device to be in connected call state before answering further calls on the same device. If calls are answered without checking for the call state of previous calls on the same device, then CTI might return a successful answer response but the call will not go to connected state and needs to be answered again. See DDTS CSCee17001 for more details.

CallInfo for Calls on QSIG Trunk

Call info on a call across a QSIG gateway is not consistent. Due to the limitations in the protocol, application would see inconsistent values for call.getLastRedirectingAddress() and call.getCalledAddress() for calls across QSIG trunks.

Refer to CSCee74730, CSCee59084, CSCee70747 and CSCsk62441 for details.

Caveats for 4.0

This section lists the JTAPI caveats for Release 4.0:

Extra Connection with Wild Card DN

JTAPI may create extra connection when Call is made wild card DN. See the release note of DDTS CSCeb57849 for scenarios.

CallInfo in Barge Scenario

CallInfo is not updated when Barge Initiator Drops the Call. See the release note of DDTS CSCec23359 for scenarios.

CallInfo Issues When Caller Redirects Call

In this scenario A calls B, A redirects to C and application is monitoring only C, the calling and called address would be B and C and the calling terminal would be terminal A.

Translation Pattern and Presentation Indication Interaction

JTAPI has getCalledAddressPI and getCurrentCalledAddressPi interfaces to receive the PIs of the originalCalled and Called parties. While making a call through a Translation Pattern, if the pattern modifies the PIs of the CalledParty, the getCalledAddressPI continues to reflect the earlier PI while the getCurrentCalledAddressPI shows the PI as set at the pattern. Refer DDTS CSCec58085 for more details.

Extra TermConnHeld Events

Applications may some times see an extra TermConnHeld Event on Controller during Transfer and Conference scenario. For transfer scenario this extra TermConnHeldEvent is sent on controller before TermConnDropped is sent. For conference scenario, for primary controller which remains in the call, this event is sent before TermConnTalking is sent, and for controller which is dropped from conference, it is sent before TermConnDropped is sent. Refer DDTS CSCec55257 for more details.

Transfer and Conference Interaction

- 9.1.6.1—in JTAPI whenever transfer is done to connect a normal call to a ConferenceCall, the GlobalCallId of Conference Call always survives irrespective of how the transfer is performed (whether on Call1 or Call2).
- 9.1.6.2—during transferring of conference scenario, it is possible External Connection creation events are delivered before CiscoTransferStartedEv, however, Call Merge events are still delivered within CiscoTransferStart and CiscoTransferEnd boundry.
- 9.1.6.3—during transferring of conference scenario, if transfer-destination is not observed by the JTAPI Application, the LastRedirectingParty is not updated. For example, if A, B, C are in conference, C (Transfer controller) transfers call to D, and D (Transfer-destination) is not observed by JTAPI, then

LastRedirectingParty of the call remains unchanged after Transfer is completed. Ideally after Transfer is completed, LastRedirectingParty is updated to Transfer controller.

Dropping a Call on Shared Lines

If there is heldCall on SharedAddress (SharedLine) and application is not observing all the terminals of SharedLine, then Connection.disconnect() using SharedAddress's connection does not drop the call. The is left with connections of OtherParty in the call. To drop the call, the application must use either Call.drop() or manually disconnect call from non observered terminals. In other words, if there is HeldCall on SharedAddress, then for the terminals that is not in Application's control, callmust be dropped manually. For detais, refer DDTS CSCed06910.

Barge Call

In a Barge Call, when

1. Barge target holds
2. Barge target does a consult conference or arbitrary Conference
3. The OtherParty drops the call
4. Barge target initiate transfer, arbitrary transfer or BlindTransfer
5. Barge target parks the call
6. Barge target Idiverts the call
7. Barge target redirects call

Then, Initiators TerminalConnection/CallCtlTerminalConnection is dropped as result of above operations. However, CallCtlCause for these TermConnDropp/CallCtlTermConnDrop would be Cause_Normal. JTAPI is unable to provide specific cause such as Cause_Redirect for #7 above.

Note Initiator is the party that presses the Barge key to barge into a call. Target is where Initiator Barges, OtherParty is address which not Initiator/Target. For details, refer DDTS CSCed07230.

Null lastRedirectingAddress

Call.getLastRedirectingAdress() returns null when only the redirecting address is observed. The call goes to INVALID state after redirect request and if application calls the above interface after redirecting the call it would see null. For details, refer CSCee92111.

Devices Configured with Same CLI

In a conference scenario, where conference controller is observed by JTAPI and other two calls are from devices configured with same the CLI (Directory Number), JTAPI creates one connection. So, when conference is completed, it has only two connections in the call instead of three. If the conference is completed using conference() API, a post condition timeout exception is thrown. For details, refer DDTS CSCeh05723.

Current Called Address

CiscoCall.getCurrentCalledAddress() returns null before the call is offered to the called address. In earlier releases, this used to return Unknown address. Applications must handle null or Unknown address returned for CiscoCall.getCurrentCalledAddress().