



Message Sequence Charts

This appendix contains the message sequence charts, which illustrate the message flows for the following scenarios:

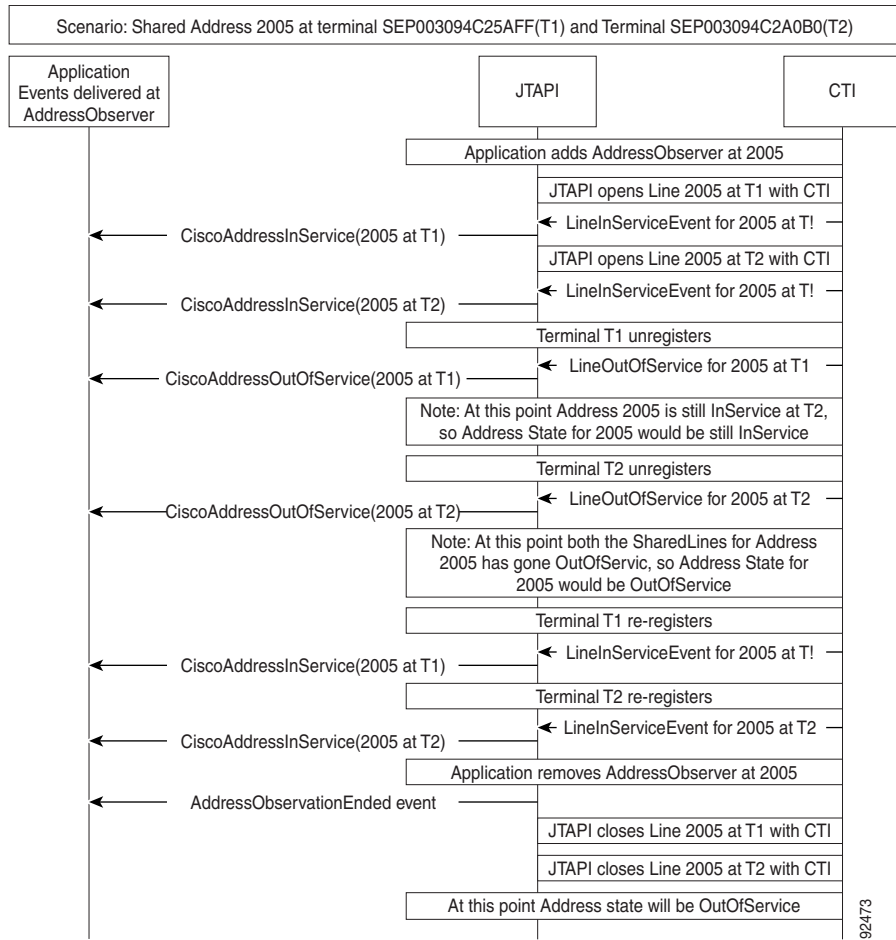
- [Shared Line Support](#)
 - [AddressInService/AddressOutOfService Events](#)
 - [Incoming Call to Shared Address](#)
 - [Outgoing Call from Shared Address](#)
 - [Shared Address Calling Itself](#)
- [Transfer and Direct Transfer](#)
 - [DirectTransfer/Arbitrary Transfer Scenario—Page 1](#)
 - [Consult Transfer Scenario](#)
- [Conference and Join](#)
 - [Join/Arbitrary Conference Scenario—Page 1](#)
 - [Consult Conference Scenario](#)
- [Barge and Privacy](#)
 - [Barge Feature](#)
 - [CBarge Feature](#)
 - [Privacy](#)
- [CallSelect and UnSelect](#)
- [Dynamic CTIPort Registration Per Call](#)
- [Media Termination at Route Point](#)
- [Redirect Set OriginalCalledID](#)
- [Single Step Transfer](#)
- [Modifying Calling Number](#)
- [AutoAccept for CTIPort and RoutePoint](#)
- [Forced Authorization and Customer Matter Codes Scenarios](#)
- [Super Provider Message Flow](#)
- [SuperProvider and Change Notification Enhancements Use Cases](#)
- [QSIG Path Replacement Use Cases](#)

- Device State Server Message Flow
- Partition Support
- Hairpin Support
- QoS Support
- TLS Security
- SRTP Key Material
- Device and Line Restriction
- SIP Support
- SIP REFER/REPLACE
- Unicode Support
- Backward Compatibility Enhancements
- Half Duplex Media Support Message Flow

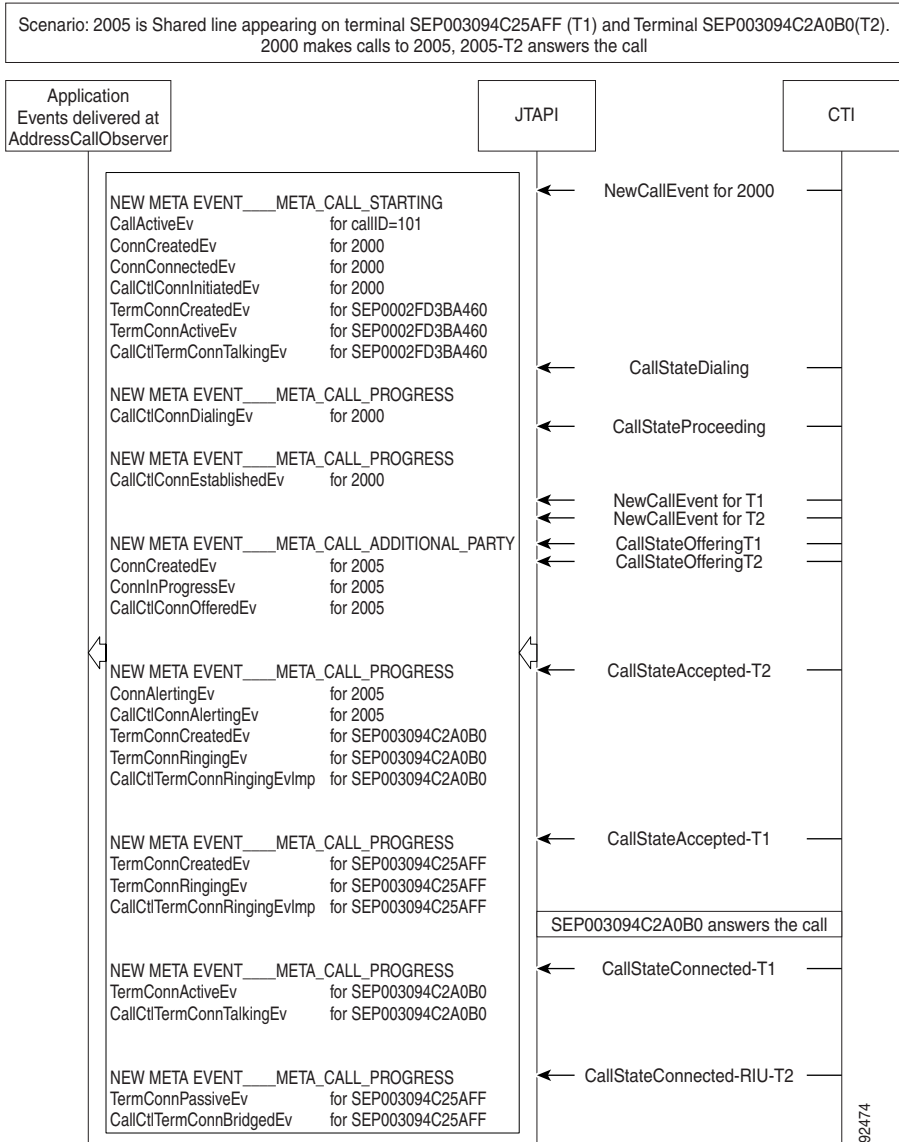
Shared Line Support

The following diagrams illustrate the message flows for Shared Line support.

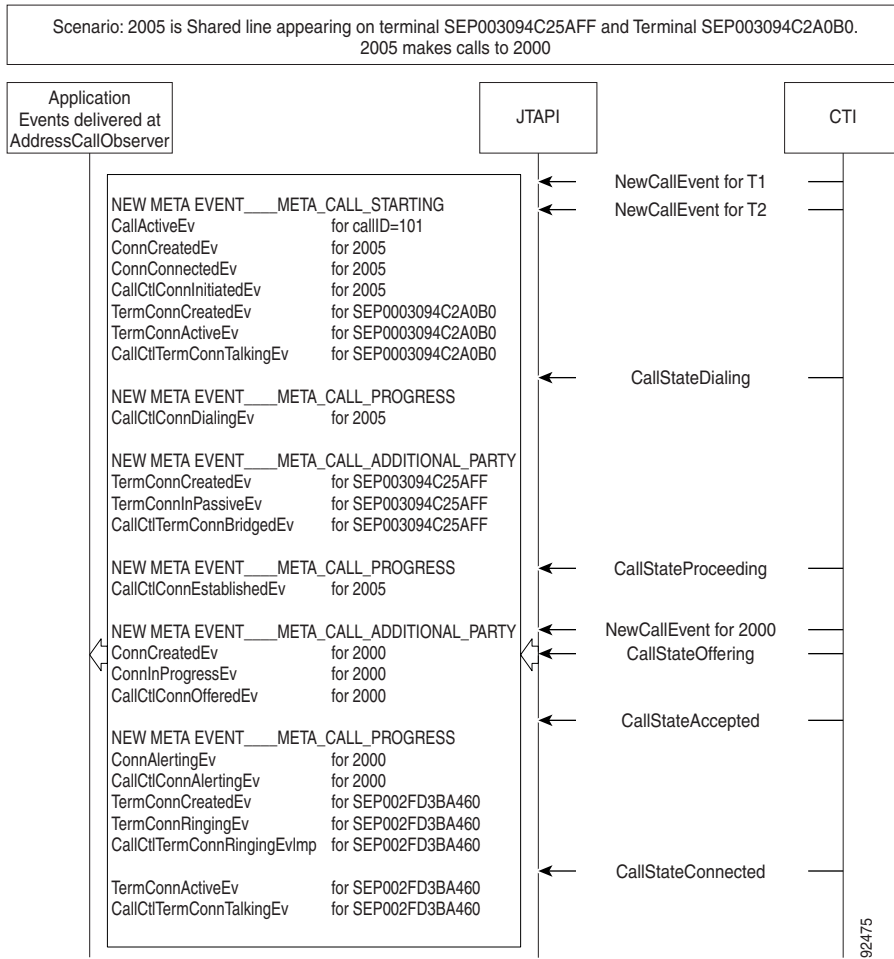
AddressInService/AddressOutOfService Events



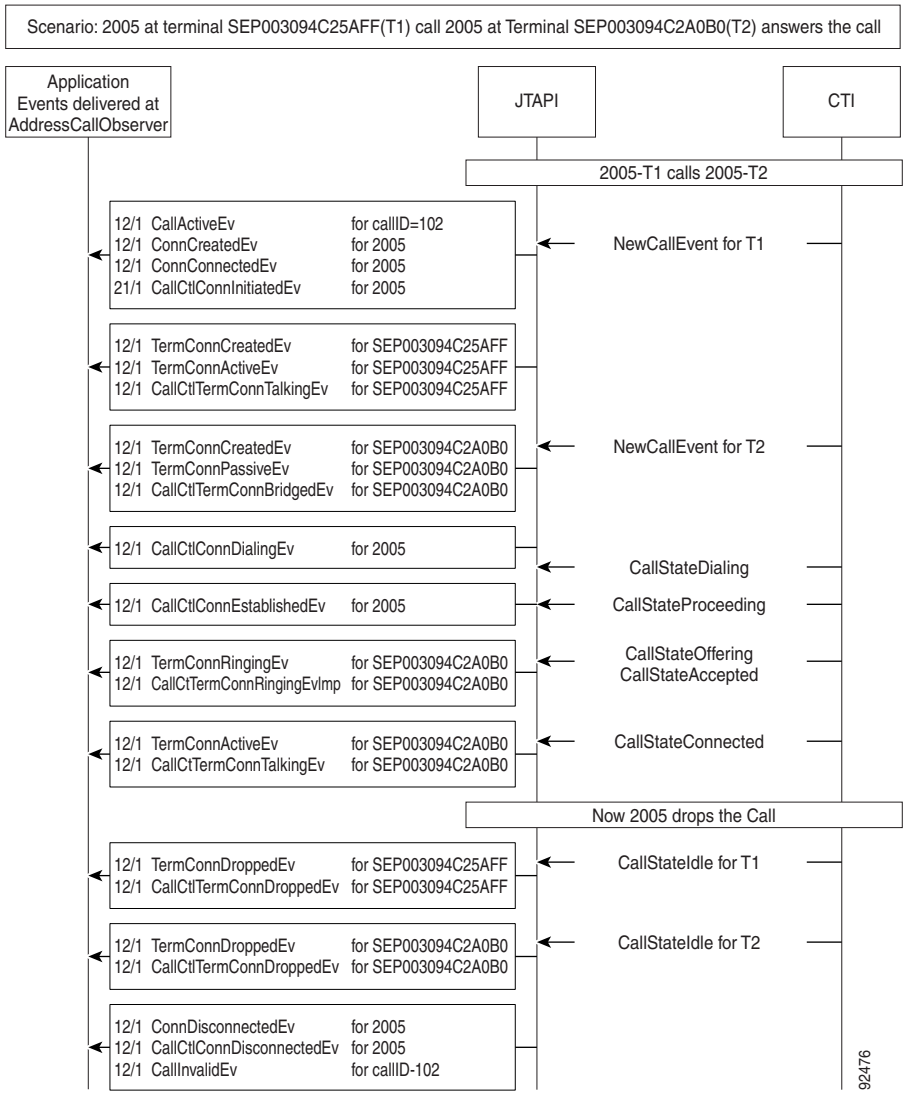
Incoming Call to Shared Address



Outgoing Call from Shared Address



Shared Address Calling Itself

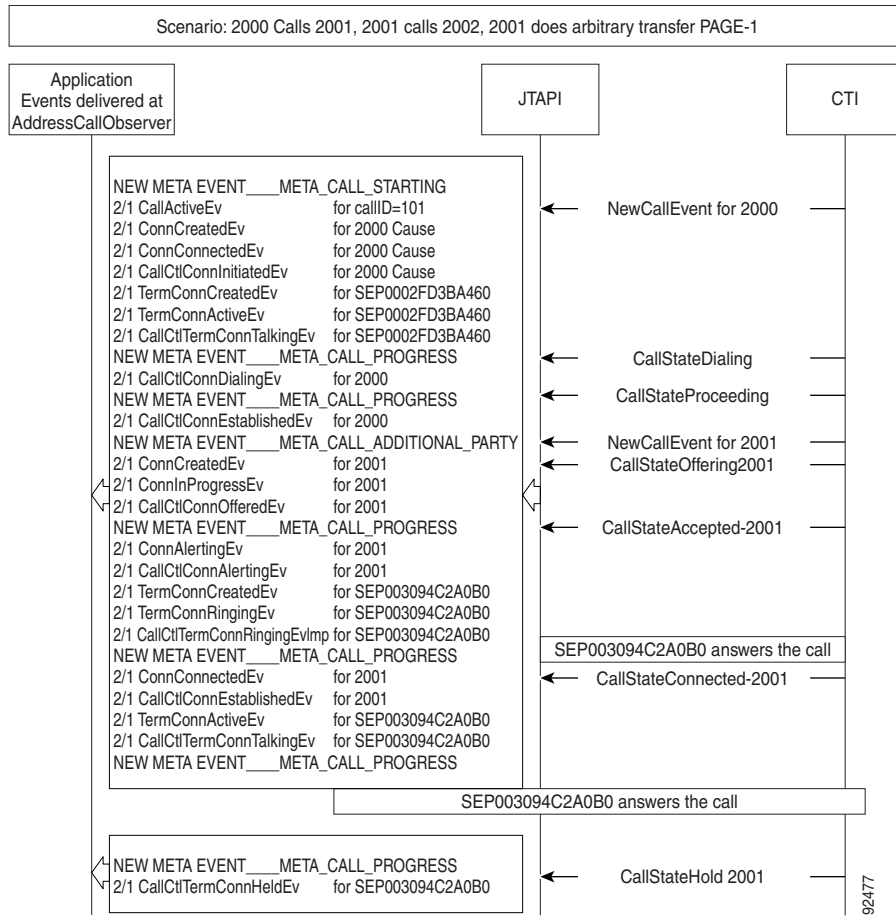


92476

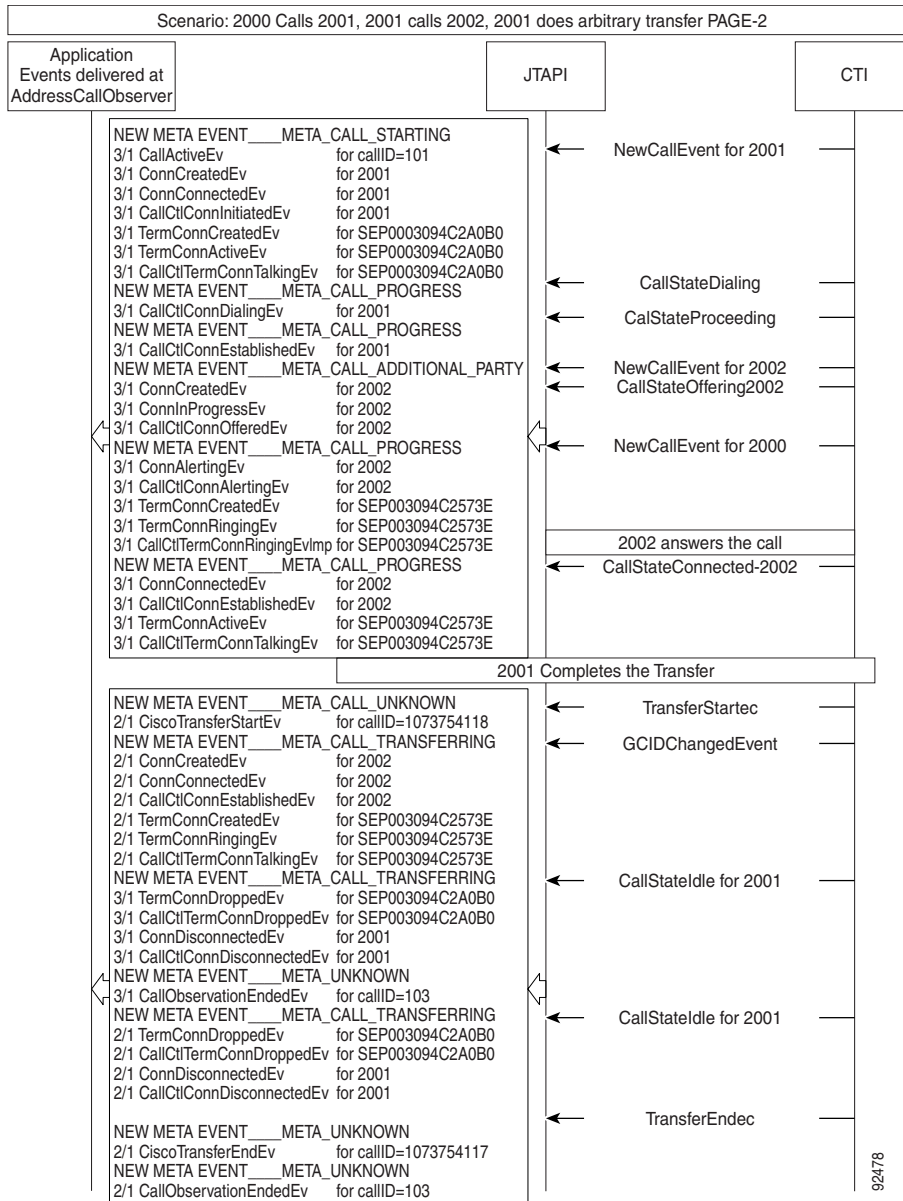
Transfer and Direct Transfer

The following diagrams illustrate the message flows for Transfer and Direct Transfer.

DirectTransfer/Arbitrary Transfer Scenario—Page 1



Direct Transfer/Arbitrary Transfer—Page 2



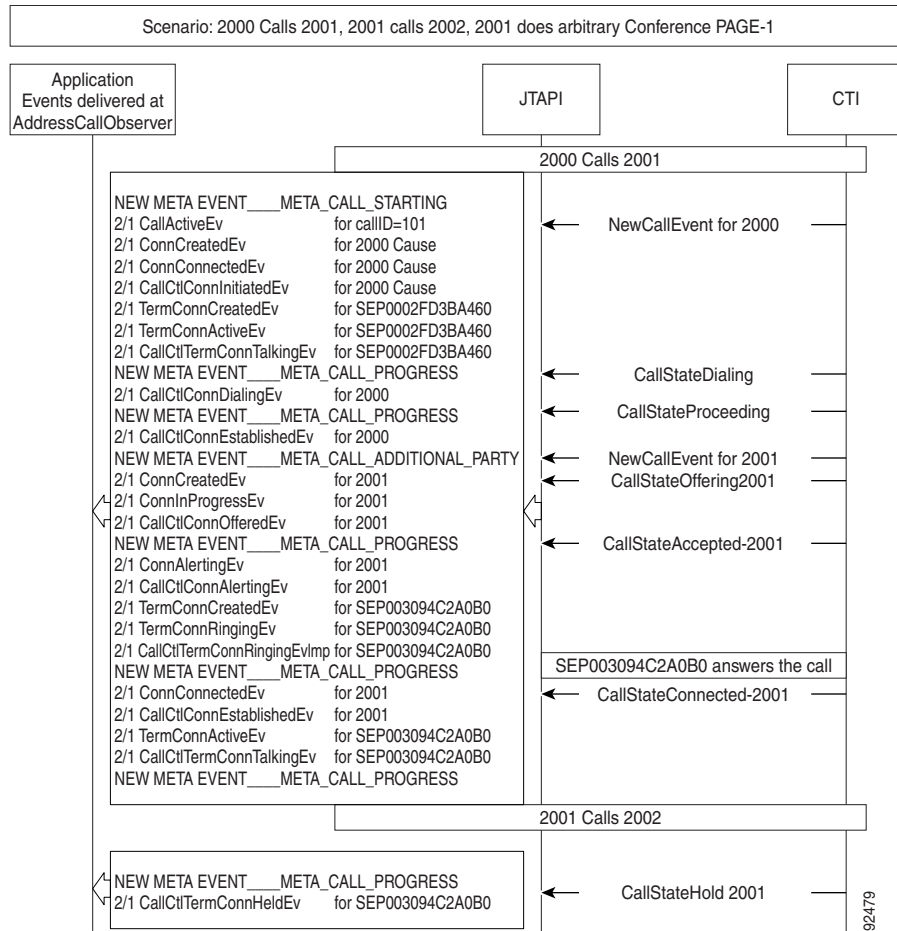
Consult Transfer Scenario

The message flow for Consult Transfer acts the same as the flow for Arbitrary Transfer.

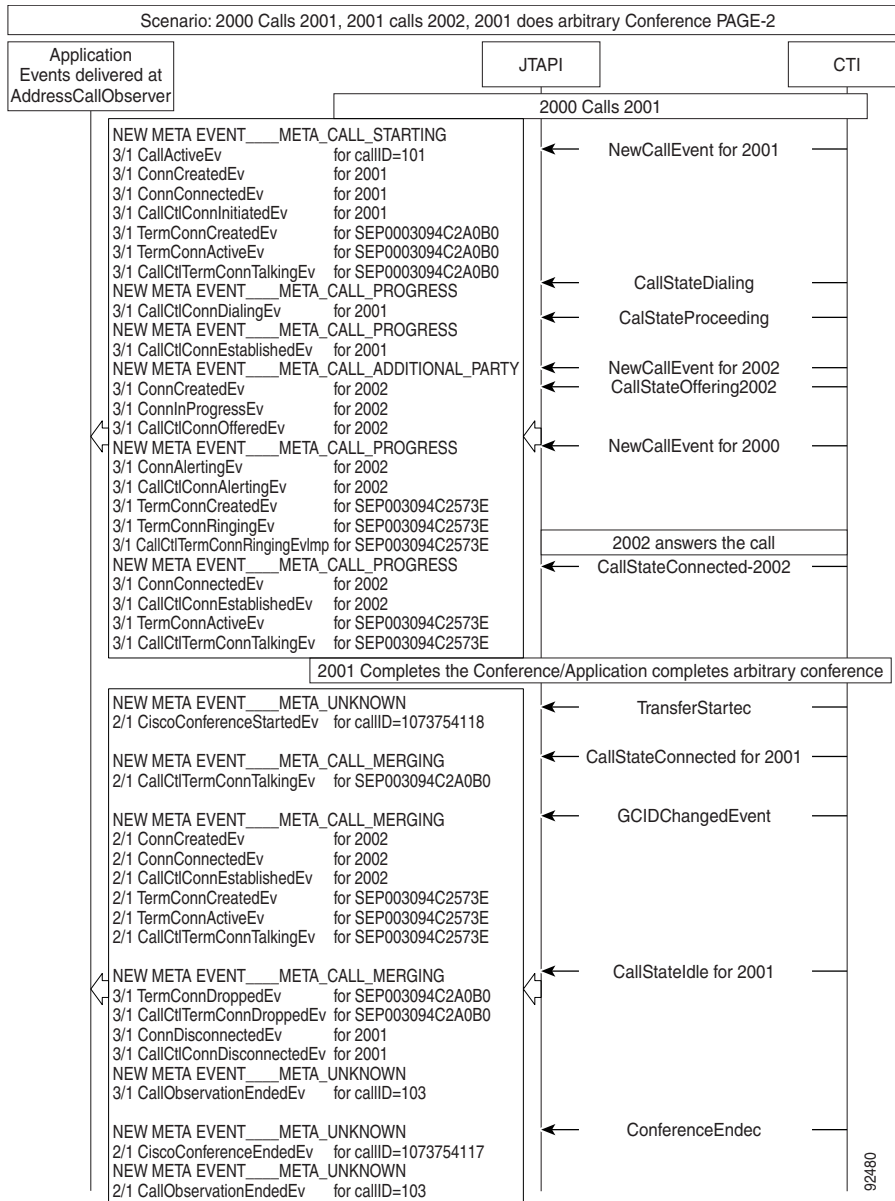
Conference and Join

The following diagrams illustrate the message flows for Conference and Join.

Join/Arbitrary Conference Scenario—Page 1



Join/Arbitrary Conference Scenario—Page 2



Consult Conference Scenario

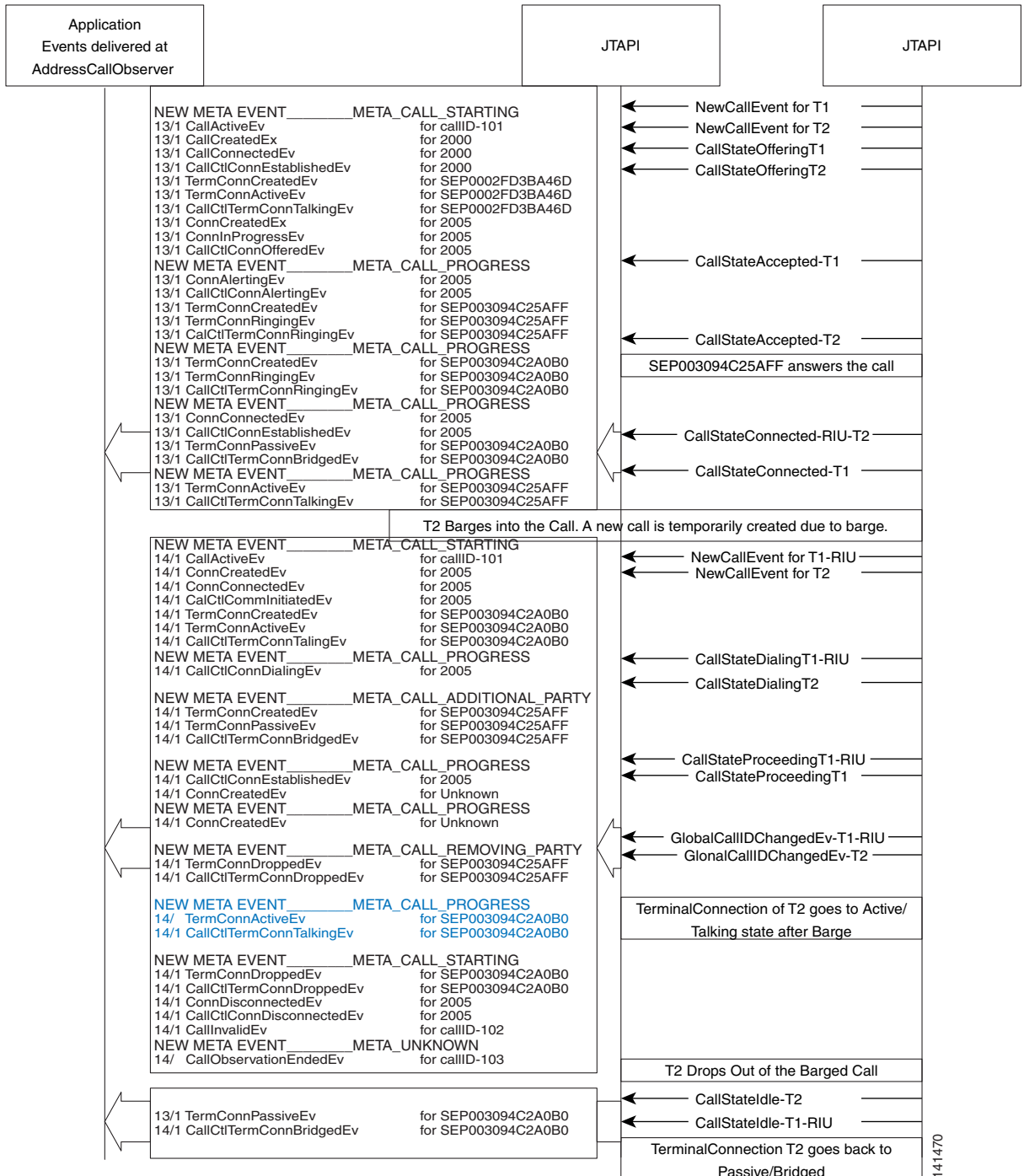
The message flow for Consult Conference acts the same as the flow for Arbitrary Conference.

Barge and Privacy

The following diagrams illustrate the message flows for Barge and Privacy.

Barge Feature

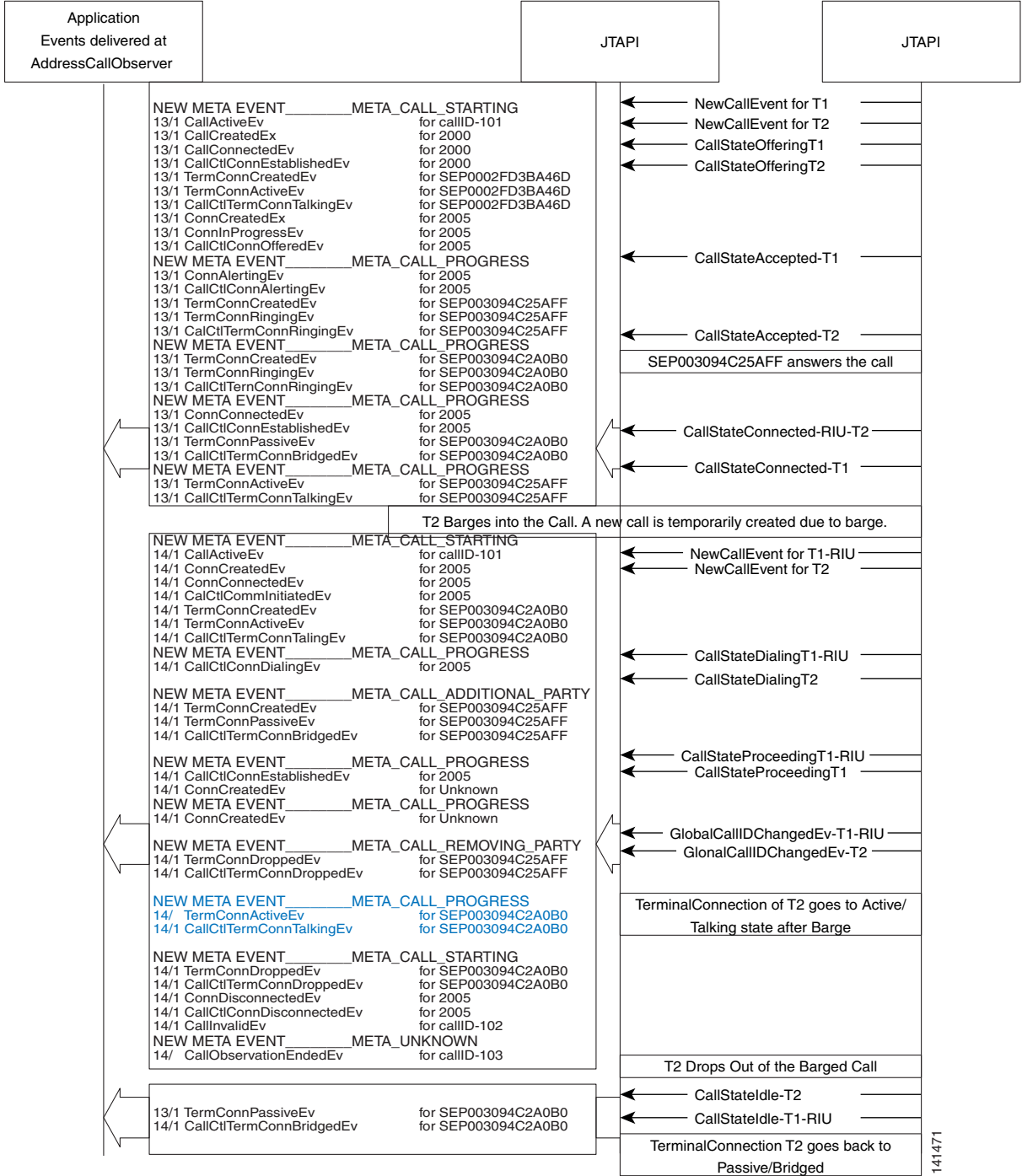
Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005, 2005-T1 answers the Call. Now T2 Barges into the Call.



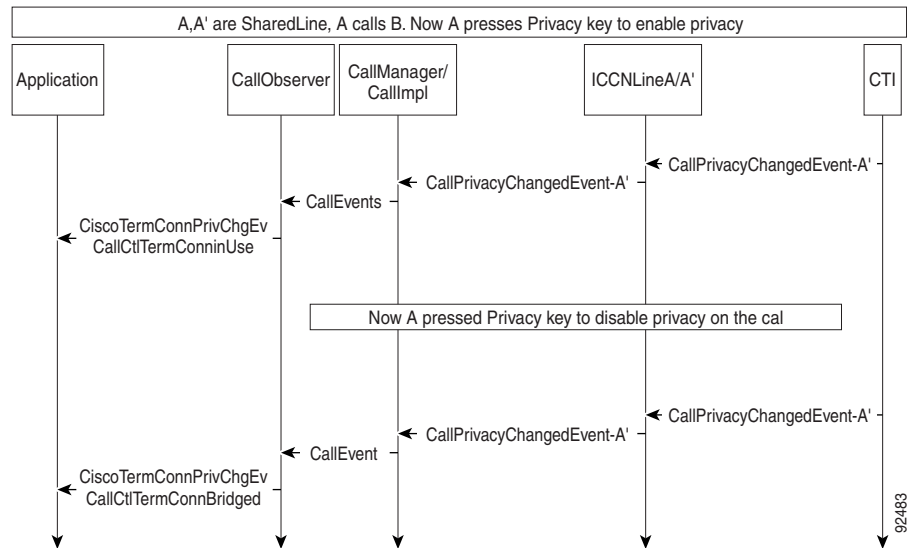
141470

CBarge Feature

Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005, 2005-T1 answers the Call. Now T2 CBarges into the Call.

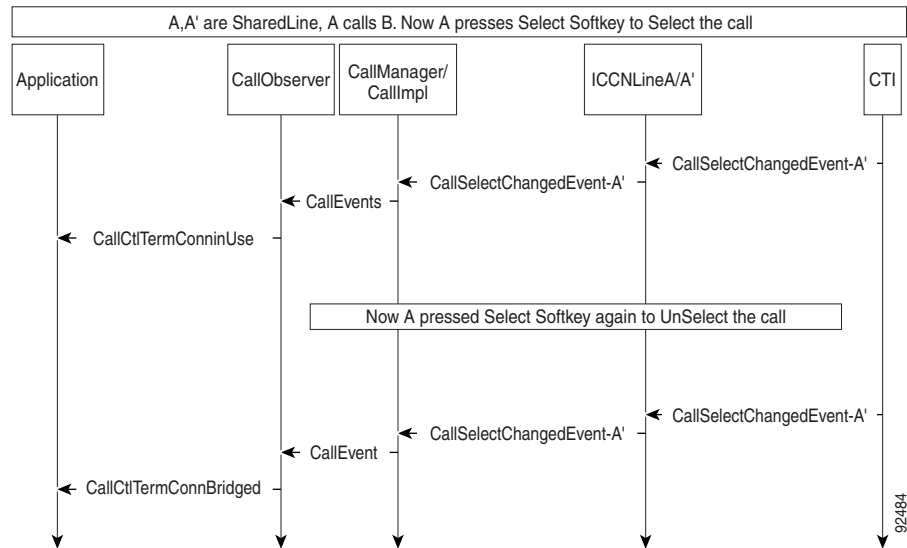


Privacy



CallSelect and UnSelect

The following diagram illustrates the message flows for CallSelect and UnSelect.

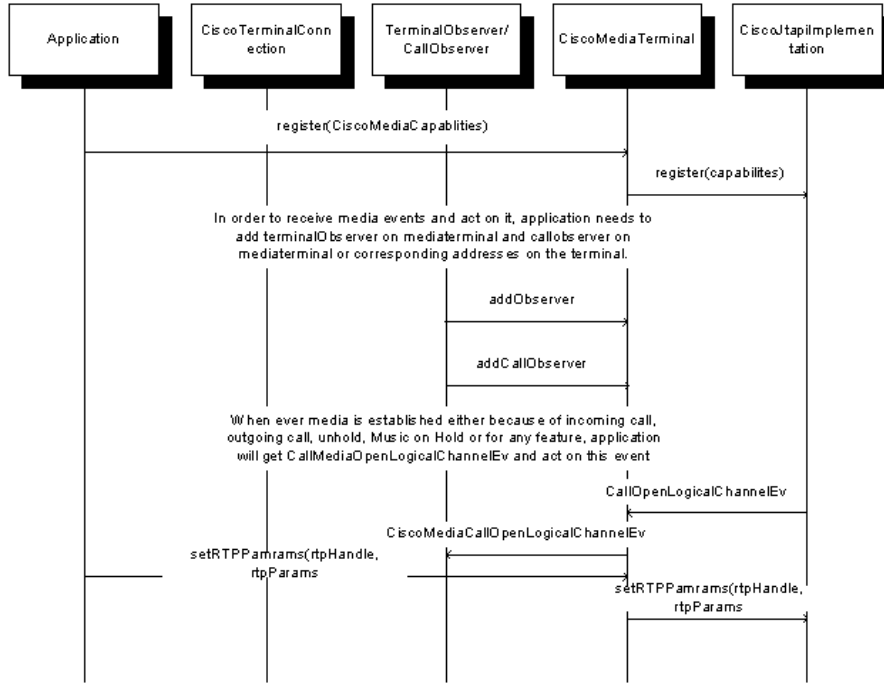


Dynamic CTIPort Registration Per Call

The following diagram illustrates the message flows for Dynamic CTIPort Registration per call.

Dynamic Registration for CTIPort

Dynamic Registration for MediaTerminal:
 In order to set ipAddress and portNo on per call basis for MediaTerminal, application needs to register MediaTerminal as specified below and need to add terminalObserver and callObserver.

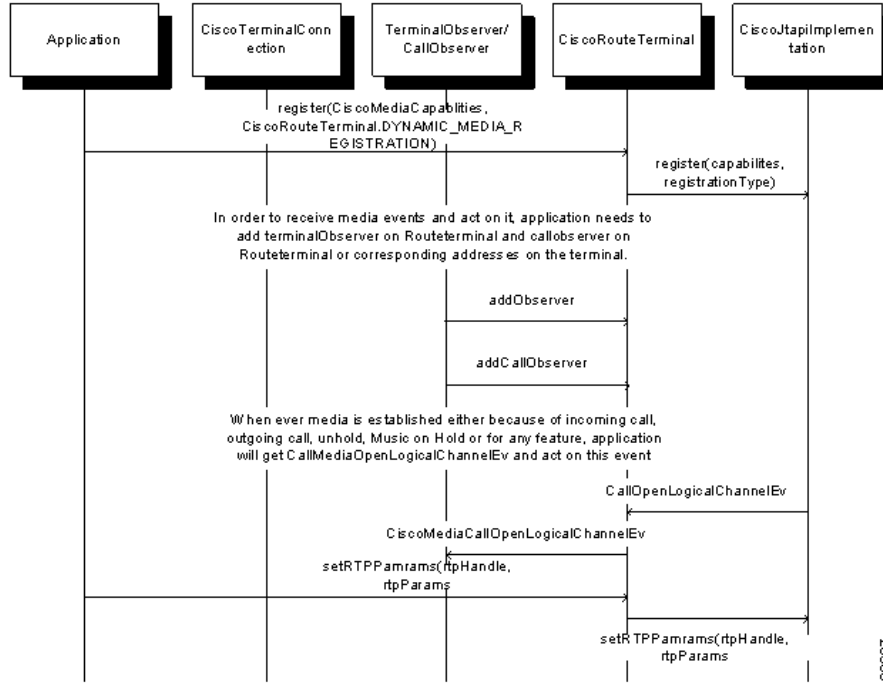


90966

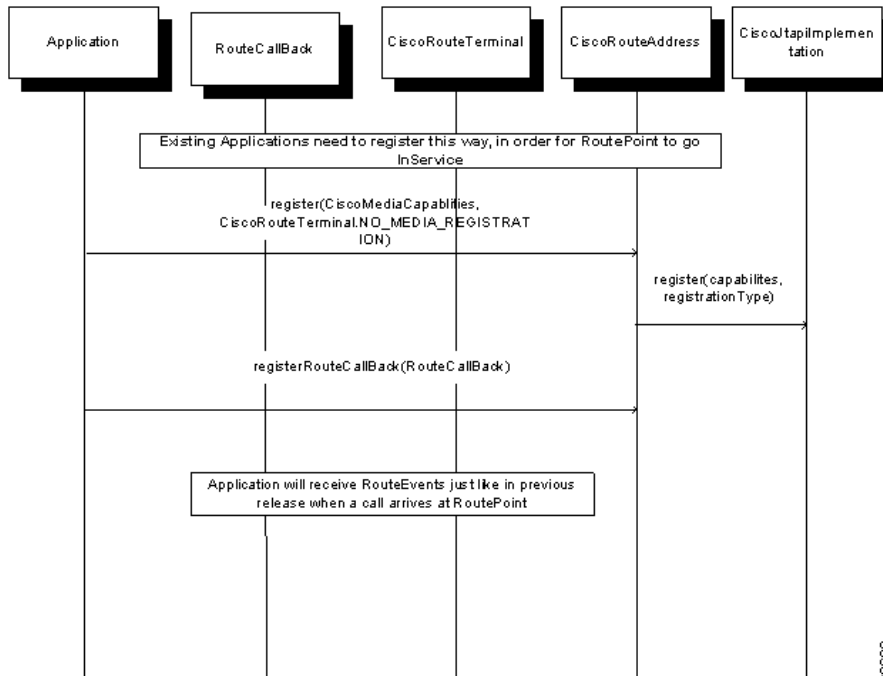
Media Termination at Route Point

The following diagrams illustrate the message flows for Media Termination at Route Point.

Media Termination at Route Point:
 In order to set ipAddress and portNo on per call basis for RoutePoint, application needs to register RouteTerminal as specified below and need to add terminalObserver and callObserver.



Media Termination at Route Point:
 Existing applications that need to use RoutePoint for pure routing, needs to register the following way.



Redirect Set OriginalCalledID

The following scenario illustrates the message flows for Redirect Set OriginalCalledID.

Scenario One

- A, B, and C appear in an applications controlled list.
- D is does not appear in the control list.
- A calls B.
- B redirects call to D with C as preferredOriginalCalledParty.

Application will see following events for parties A and B:

Meta Event Cause	Call	Event	Fields
META_CALL_ADDING_PARTY	Call 1	ConnCreatedEv for D ConnConnectedEv for D CallCtlConnEstablishedEv for D	CallingParty=A CalledParty = B LastRedirectedParty=C CurrentCalledParty=D
META_CALL_REMOVE_PARTY	Call 1	ConnDisconnectedEv for B CallCtlConnDisconnectedEv for B TermConnDroppedEv for B CallCtlTermConnDroppedEv for B CallObservationEndedEv for B	CallingParty=A CalledParty = B LastRedirectedParty=C CurrentCalledParty=D



Note

The specified event group may not be in the same order and might change depending on where parties are present in the cluster, on the load, and other conditions.

Scenario Two

- A, B, and C do not appear in the Control list, and
- D is in the application control list.
- A calls B.
- B redirects the call to D with C as preferredOriginalCalledParty.

The application will see following events for party D:

Meta Event Cause	Call	Event	Fields
META_CALL_STARTING	Call 1	CallActiveEv ConnCreatedEv for D ConnInProgressEv for D CallCtlConnOfferedEv for D ConnCreatedEv for A CallCtlConnInitiatedEv for A	CallingParty=A CalledParty = D LastRedirectedParty=C CurrentCalledParty=D
META_CALL_PROGRESS	Call 1	ConnAlertingEv for D CallCtlConnAlertingEv for D TermConnCreatedEv for D CallCtlTermConnRingingEv for D ConnConnectedEv for A CallCtlConnEstablishedEv for A	CallingParty=A CalledParty = D LastRedirectedParty=C CurrentCalledParty=D

Meta Event Cause	Call	Event	Fields
META_CALL_PROGRESS	Call	ConnConnectedEv for D CallCtlConnEstablishedEv for D TermConnActiveEv for D CallCtlTermConnTalkingEv for D	CallingParty=A CalledParty = D LastRedirectedParty=C CurrentCalledParty=D

Single Step Transfer

Addresses A, B, and C appear in the control list, and the call between A and B is then gets transferred to C with B as the transfer controller. Applications will see the following events:

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string)	ConnCreatedEv 5003 Cause: CAUSE_NORMAL ConnInProgressEv 5003 Cause: CAUSE_NORMAL CallCtlConnOfferedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnAlertingEv 5003 Cause: CAUSE_NORMAL CallCtlConnAlertingEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	NEW META EVENT__ META_CALL_REMOVING_PARTY TermConnDroppedEv CTIP2 Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv CTIP2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnDisconnectedEv 5002 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5002 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv 5003 Cause: CAUSE_NORMAL ConnInProgressEv 5003 Cause: CAUSE_NORMAL CallCtlConnOfferedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER ConnCreatedEv 5001 Cause: CAUSE_NORMAL ConnConnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnEstablishedEv 5001 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string) (continued)	CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 5003 CAUSE_NORMAL CallCtlConnEstablishedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL	NEW META EVENT_____META _UNKNOWN CallObservationEndedEv Cause: CAUSE_NORMAL	ConnAlertingEv 5003 Cause: CAUSE_NORMAL CallCtlConnAlertingEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv CTIP3 TermConnRingingEv CTIP3Cause: CAUSE_NORMAL CallCtlTermConnRinging EvImpl CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 2004 Cause: CAUSE_NORMAL CallCtlConnEstablishedEv 5003Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string) (continued)			TermConnActiveEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnTalking Ev CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStoppedEv Cause: CAUSE_NORMAL CiscoRTPOutputStopped Ev Cause: CAUSE_NORMAL ConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnDisconnected Ev 5001 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnDroppe dEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL ConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallCtlConnDisconnected Ev 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL META_UNKNOWN CallInvalidEv [#32] Cause: CAUSE_NORMAL

Modifying Calling Number

The following scenario illustrates the message flows for Modifying Calling Number.

Scenario One

The application controls the device Route Point (RP) and registers RP .

A and B are PNO and appear within the Cisco Unified CallManager cluster.

A calls RP.

Call arrives at RP

Action	Event	Fields
Call Arrives at RP	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A)
Application invokes selectRoute(routeselected[], callingsearchspace, modifyingcallingnumber[]) where routeSelected[] = C callingSearchSpace = CiscoRouteSession.DEFAULT_ SEARCH_SPACE	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A) getRouteUsed () = C
Application invokes endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_END getRouteAddress () = RP

Scenario Two

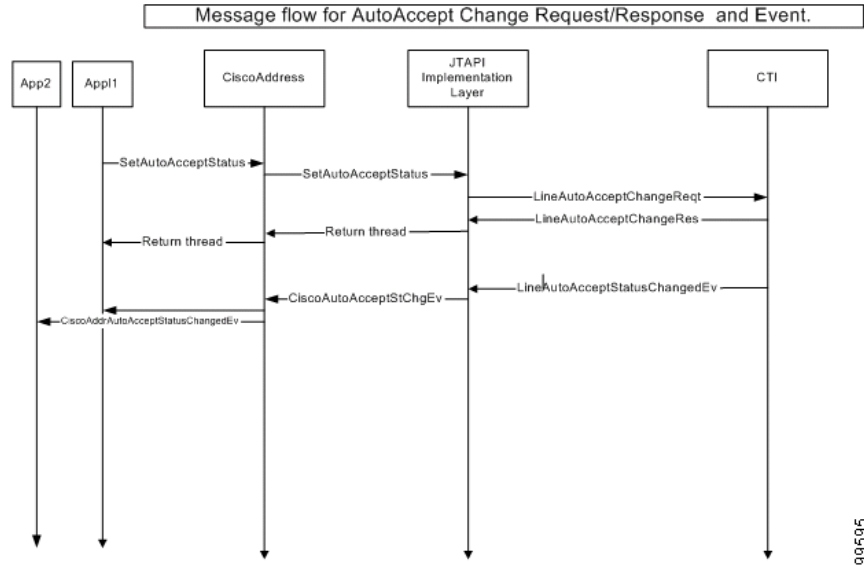
The application is controls A and B.

A calls RP, which selectsRoute call to B with modified calling number as M.

Action	Event	Fields
A calls RP, which is not in controlled list.	NEW META EVENT_____META_CALL_ STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_ PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_ PROGRESS CallCtlConnEstablishedEv A ConnCreatedEv RP ConnInProgressEv RP CallCtlConnOfferedEv RP	getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= getModifiedCallingAddress()=A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= getModifiedCallingAddress()=A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()=A getModifiedCalledAddress() =B

Action	Event	Fields
Another application controls the RP selectRoute to B with modifying calling number as M.	NEW META EVENT_____META_CALL_ ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnDisconnectedEv RP CallCtlConnDisconnectedEv RP	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= M getModifiedCalledAddress() =B
	NEW META EVENT_____META_CALL_ PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP
	TermConnRingingEv B CallCtlTermConnRingingEv B	getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= M getModifiedCalledAddress() =B
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= M getModifiedCalledAddress() =B

AutoAccept for CTIPort and RoutePoint



Forced Authorization and Customer Matter Codes Scenarios

Scenario One

The application controls A and B; B requires a forced authorization code (FAC) to extend the call.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtiConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtiTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtiConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. FAC_REQUIRED

Action	Event
Application enters additional digits by using CiscoConnection.addToAddress .	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B ConnConnectedEv B CallCtlConnEstablishedEv B
B answers the call.	TermConnActiveEv B

Scenario Two

The application controls A and B; B requires both an FAC and a CMC (client matter code) to extend the call.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. FAC_CMC_REQUIRED
Application enters FAC code digits with # termination by using CiscoConnection.addToAddress within the T302 timer.	NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. CMC_REQUIRED

Action	Event
Application enters CMC code digits with # terminated by using CiscoConnection.addToAddress within T302 timer.	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B

Scenario Three

The application controls A and B;

B requires a CMC, and the application enters an invalid code.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. CMC_REQUIRED

Action	Event
The application enters the incorrect CMC digits (# terminated) by using CiscoConnection.addToAddress within the T302 timer limit.	NEW META EVENT _____ META_CALL_PROGRESS ConnFailedEv A CallCtlConnFailedEv A getCiscoCause () = CiscoCallEv.FAC_CMC
The application receives reorder tone.	NEW META EVENT _____ META_CALL_ENDING TermConnDroppedEv CallCtlTermConnDropped ConnDisconnectedEv CallCtlConnDisconnectedEv CallInvalidEv CallObservationEndedEv

Scenario Four

The application controls both A and B; A calls B; B redirects the call to C, which needs both an FAC and a CMC.

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT _____ META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT _____ META_CALL_PROGRESS CallCtlConnDialingEv A NEW METAEVENT _____ META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT _____ META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv SEPB TermConnRingingEv SEPB CallCtlTermConnRingingEv SEPB ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv SEPB CallCtlTermConnTalkingEv SEPB

Action	Event
B issues a redirect request to C and passes an FAC and a CMC code.	<p>NEW META EVENT_____META_CALL_REMOVING_PARTY TermConnDroppedEv SEPB CallCtlTermConnDroppedEv SEPB Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnDisconnectedEv B Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED</p> <p>NEW META EVENT_____META_CALL_PROGRESS ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED</p> <p>NEW META EVENT_____META_CALL_PROGRESS ConnInProgressEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED</p> <p>NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv A Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED</p> <p>NEW META EVENT_____META_CALL_PROGRESS ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR</p>

Scenario Five

Application controls the device Route Point (RP) and registers the RP.

A and B are PNO and within the Cisco Unified CallManager cluster.

Action	Event	Fields
Call arrives at RP	RouteEvent	State = ROUTE getRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A)
Application invokes selectRoute(routeselected[], callingsearchspace, modifyingcallingnumber[], preferredOriginalCdNumber[], preferredOriginalCdOption[], facCode[], cmcCode[]) where routeSelected[] = B callingSearchSpace = CiscoRouteSession.DEFAULT_ SEARCH_SPACE modifyingCgNumber = null, preferredOriginalCdNumber = null, preferredOriginalCdOption = CiscoRouteSession.DONOT_R ESET_ORIGINALCALLED, facCode[] = "facCode for B" cmcCode[] = "cmcCode for B"	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A) getRouteUsed () = B
Application invokes endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_END getRouteAddress () = RP

Super Provider Message Flow

The application tries to create Terminal for CTIPort1 that has Addresses 2000 and 2001. The following events get sent to the application.

No.	Action	Event
1	Application invokes CiscoProvider.CreateTerminal(CTIPort1) where CiscoProviderCapabilities. canObserveAnyTerminal() returns TRUE.	JTAPI would return CiscoTerminal object and the following events get sent: CiscoTermCreatedEv CTIPort1<----- CiscoAddrCreated 2000<----- CiscoAddrCreated 2001<-----
2	If the application already has a terminal where the 2001 address already exists, that is, 2001 is a SharedLine Address. Now, the application invokes CiscoProvider.CreateTerminal(CTIPort1)	JTAPI would return CiscoTerminal object and the following events get sent CiscoTermCreatedEv CTIPort1<----- CiscoAddrCreated 2000<----- CiscoAddrAddedToTerminalEv 2001<-----
3	Application invokes CiscoProvider. CreateTerminal(CTIPortX) where CTIPortX does not exist in Cisco Unified CallManager cluster.	JTAPI would throw an exception: InvalidArgumentException
4	Application invokes CiscoProvider. CreateTerminal(CTIPort1) where CiscoProviderCapabilities.canObserve AnyTerminal() returns FALSE.	JTAPI would throw an exception: PrivilegeViolationException

SuperProvider and Change Notification Enhancements Use Cases

New events have been added to JTAPI, which will be sent to applications in order to handle new failover scenario and change notification. This enhances JTAPI's ability to handle failover scenarios and the time required to shift between Superprovider and normal user modes.

Scenario A

Superprovider user opens provider and opens a few devices in Superprovider mode which are not in control list. From admin pages, Superprovider privilege is removed.

Expected Behavior

Application receives CiscoProviderCapabilityChangedEvent event. JTAPI sends CiscoTermRemovedEv all the devices which are opened / acquired and are not in the control list. JTAPI will send provider OOS to application, CiscoTermRemovedEv to devices not in control list and will reopen connection to CTI. When connect succeeds, JTAPI will send provider in service event to the app. Else, it will close the provider.

Scenario B

Normal user opens provider and opens a few devices in control list. From admin pages, Superprovider privilege is added to the user.

Expected Behavior

Application receives `CiscoProviderCapabilityChangedEvent` event. User will now be able to acquire/open devices not in its control list.

Scenario C

Normal user opens provider and opens a few park DNs. From admin pages, park DN monitor privilege is removed for the user.

Expected Behavior

Application receives `CiscoProviderCapabilityChangedEvent` event. JTAPI will cleanup all park DN addresses.

Scenario D

Normal user opens provider. From admin pages, park DN monitor privilege is added for the user.

Expected Behavior

Application receives `CiscoProviderCapabilityChangedEvent` event. Application registers the park DN monitoring feature and is able to monitor park DN.

Scenario E

Normal user opens provider. From admin pages, “modify calling party” privilege is removed for the user.

Expected Behavior

Application receives `CiscoProviderCapabilityChangedEvent` event. Application is not able to change the calling party number during redirect. JTAPI will throw error if application tries to do this.

Scenario F

Normal user opens provider. From admin pages, “modify calling party” privilege is added for the user.

Expected Behavior

Application receives `CiscoProviderCapabilityChangedEvent` event. Application is able to change the calling party number in a call during redirect.

Scenario G

Superprovider user opens provider and acquires a device not in control list. From admin pages, the device is deleted.

Expected Behavior

Application receives `CiscoTermRemovedEv` event. Device is closed from JTAPI perspective.

QSIG Path Replacement Use Cases

The following table shows the JTAPI events that are delivered to applications when calls between PBXs that are connected by Q.Signaling (QSIG) trunks are transferred and forwarded. This table also shows the events that are delivered to applications when the real-time path (RTP) is optimized by the QSIG Path Replacement feature.

Calls going out on a QSIG trunk may not have a connection for the far end if any translation pattern is changing the pattern. In other words, when the application sees two calls in the trombone case, B may not serve as the common connection on the calls.

No.	Action	Event
1	<p>A registered with CM1, B is registered with CM2, and C registered with CM3.</p> <p>A calls B (GC1); B transfers the call to C. The application is monitors C. The PR feature replaces the path after the call gets connected to C.</p> <p>The same action applies to scenarios that involve call forward at B. (The called party transfers the call.)</p>	<p>These events get delivered to applications:</p> <p>CallCtrlConnectionEstablishedEv A</p> <p>CallCtrlConnectionDisConnectEv B</p> <p>OpenLogicalChannelEvent if C is a CTI device (Dynamically registered CTIPorts and RP)</p>
2	<p>A registered with CM1, B registered with CM2, and C registered with CM3. B calls C; C answers; B transfers the call to A. A answers. The application is monitors only C. (The calling party transfers the call.)</p>	<p>In this case, both A and C represent called parties when transfer completes. After the call is answered, PR replaces the path. In this case, A and C represent IP phones; the display will be updated as a part of PR feature operation, that makes either A or C as calling.</p> <p>JTAPI events:</p> <p>GC1: CallCtlConnEstablishedEv A</p> <p>GC1: CallCtlConnDisconnectedEv B</p>

No.	Action	Event
3	3.Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. A calls B (GC1); B answers and transfers the call to C (GC2). Path replacement connects A and C bypassing CM2. The application observes both A and C. (The called party transfers the call.)	<p>For GC1 Call Observer:</p> <p>GC1: CallCtlConnEstablishedEv C</p> <p>GC1: CallCtlConnDisconnected B</p> <p>Before the PR feature replaces the path, the application sees two calls: GC1 with connections to A and C (external) and GC2 with connections to C and A(external).</p> <p>When the PR feature replaces the path, either GC1 changes GC2, or GC2 changes to GC1.</p> <p>Assuming A's GCID changes from GC1 to GC2:</p> <p>GC1: CiscoCallChangedEv (oldGCID=GC1,newGCID=GC2)</p> <p>GC1: CallCtlConnDisconnected for A</p> <p>GC1: CallCtlConnDisconnected for C</p> <p>GC1: CallInValid</p> <p>GC2: TermConnTalkingEvent for TerminalA cause= CAUSE_QSIG_PR</p>
4	Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. B calls A and transfers the call to C. Path replacement connects A and C, bypassing CM2. Application observes both A and C. (The calling party transfers the call.)	<p>Before the PR feature replaces the path, the application will see two calls: GC1 with connections to A and B (external) and GC2 with connections to C and B(external). In this case, the application will not see any transfer start events.</p> <p>When PR feature replaces the path, it updates the display of A and C and path gets replaced, resulting in a GCID change. Assuming A's GCID is changed and made the calling party, the following JTAPI events occur:</p> <p>GC1: CiscoCallChangedEv (GC1 to GC2)</p> <p>GC1: CallCtlConnDisconnected for A</p> <p>GC1: CallCtlConnDisconnected for C</p> <p>GC1: CallInValid</p> <p>GC2: ConnCreatedEv A</p> <p>GC2: ConnConnectedEv A</p> <p>GC2: TermConnTalkingEvent for TerminalA cause= CAUSE_QSIG_PR</p>
5	A registered to CM1, B registered to CM2, and C registered to CM1. A calls B; B transfers the call to C. C answers and before path replacement completes, C invokes a feature (park, redirect, and so on).	Path replacement gets abandoned.

No.	Action	Event
6	In some conditions, call processing ignores feature requests (redirect, park, transfer, and so on). This happens when a setup request is sent out, and the local CM is waiting for connect from the other end.	JTAPI: Exception will be thrown from JTAPI for feature requests.
7	In some cases, the application could receive dead air when CM goes down when the PR feature is trying to switch the RTP path. This could happen to a previously connected call.	No events JTAPI Apps: Hang up the call

Device State Server Message Flow



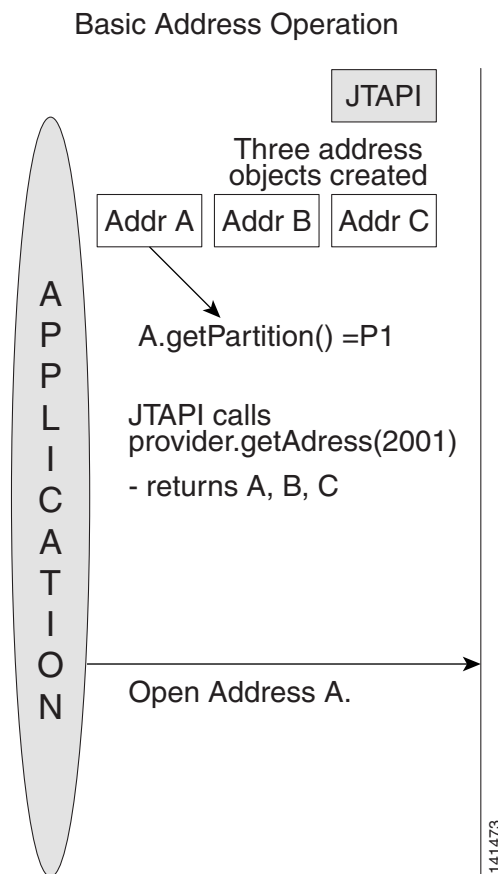
Partition Support

Since the address hashing mechanism in JTAPI has changed, this feature is expected to have performance degradation in address lookup time and during load tests.

Using getPartition() API

The example given below illustrates how getPartition(), will be used by JTAPI and applications, to differentiate between addresses having same DN but belonging to different partitions.

Example 1 Using getPartition() API



In this case, there are three addresses which belong to three different partitions: A(2001, P1), B(2001, P2) and C(2001, P3), where 2001 indicates DN and P1, P2, and P3 denote different partitions.

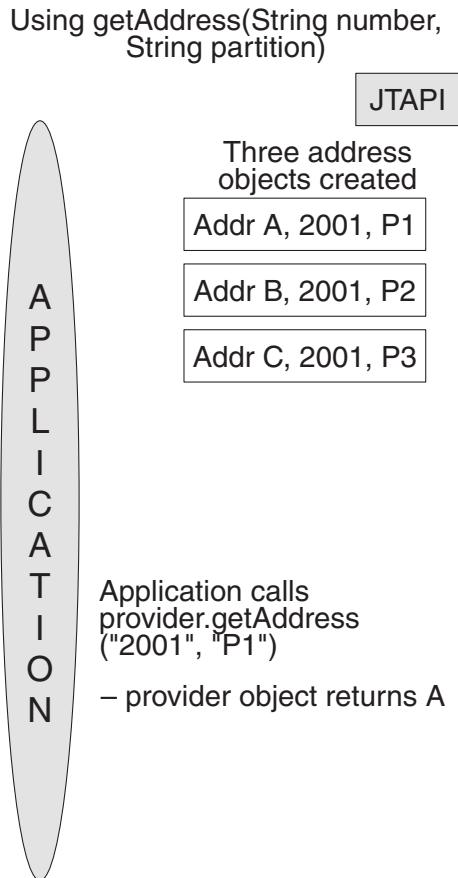
When JTAPI calls provider.getAddress("2001"), the provider object will return an array of three address objects containing A, B and C, since all of them have the same DN.

The application and JTAPI will distinguish between the three addresses by using the getPartition() method of the address object.

Using getAddress(String number, String partition) API

Consider the example shown below to see how JTAPI will use the getAddress(String number, String partition) API to retrieve the address object corresponding to a particular DN and partition when there are multiple addresses with same DN and different partitions.

Example 2 Using getAddress(String number, String partition) API



In this case, there are three addresses which belong to three different partitions. Let us denote them by A(2001, P1), B(2001, P2) and C(2001, P3), where 2001, indicates DN and P1,P2,P3 denote different partitions.

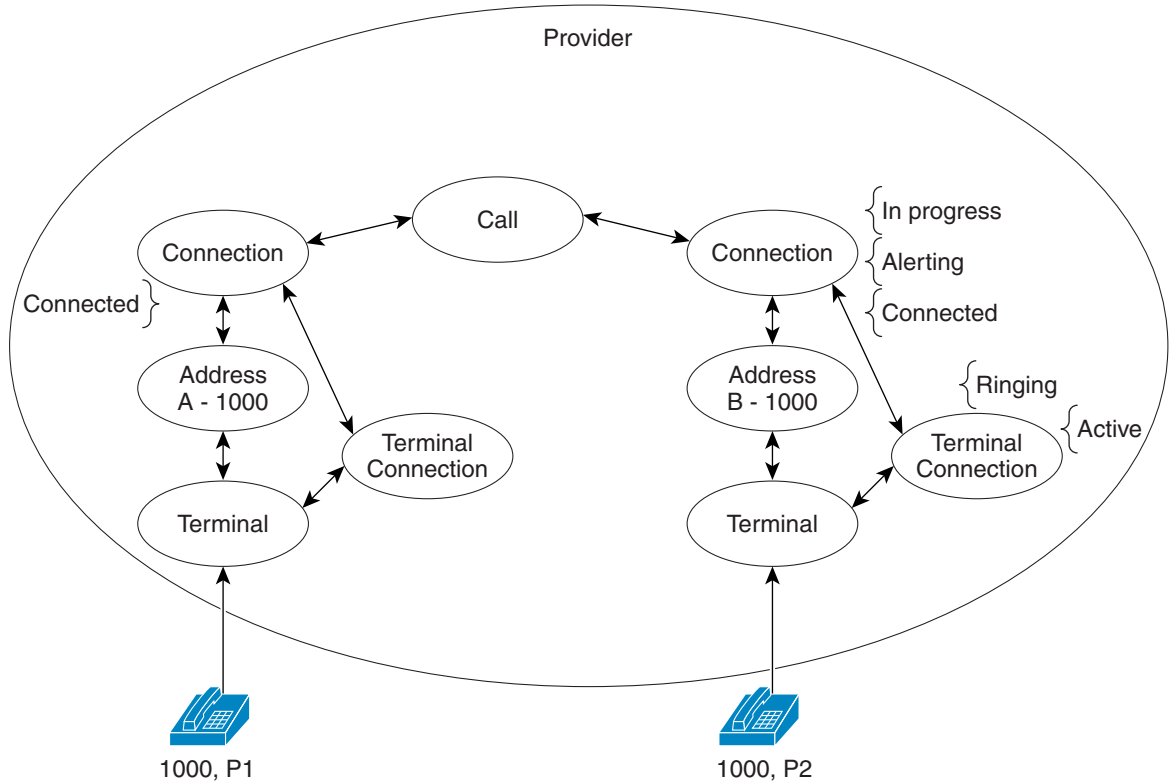
When JTAPI calls *provider.getAddress("2001", "P1")*, the provider object will return the address object which has the same DN i.e. 2001 and the same partition info, that is P1—as provided in the API. In this case, the address object A will be returned to the application.

Simple Call Scenario

Consider the following scenario where A calls B. A has DN 1000 and calls B which also has DN 1000. A belongs to partition P1 and B belongs to partition P2. The following diagram illustrates the various events and the results of API calls pertaining to this scenario, which are relevant to partition support feature.

Example 3 Simple Call Scenario

Scenario: Call from line x1000 :P1" to line x1000 "P2"



- Provider → getAddress(1000) - A and B
- Provider → getAddress(1000, "P1") - A
- Call → getCurrentCallingAddress() - A
- Call → getCurrentCalledAddress() - B
- Address A - 1000 → getPartition () - A

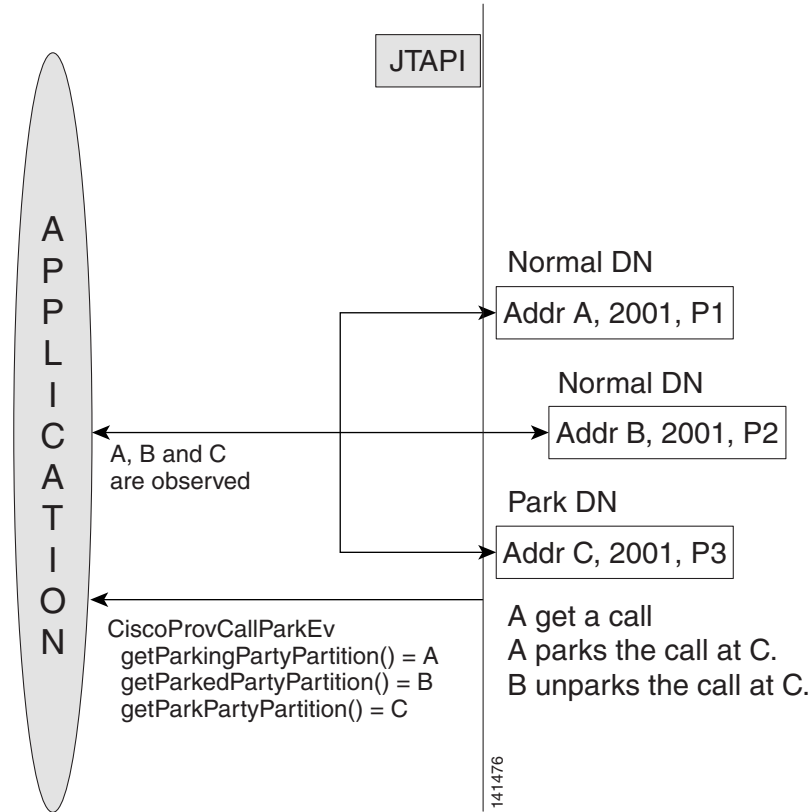
141475

Park DN Scenario

Park DNs are also treated as addresses in JTAPI. Hence, the same treatment given to normal DN is also given to park DN. The following message flow illustrates how an application will use park DN partition information in a call where park DNs are involved.

Example 4 Park DN Scenario

CiscoProvCallParkEv - API Usage



When the application is monitoring park DN, it is possible to have the park DN to be the same as a regular DN (while both belong to different partitions).

In this case, C is a park DN having same DN value as A and B while belonging to a different partition.

A receives a call and parks the call at C. B un parks the call. While the call is parked, and un parks, CiscoProvCallParkEv is generated. The API

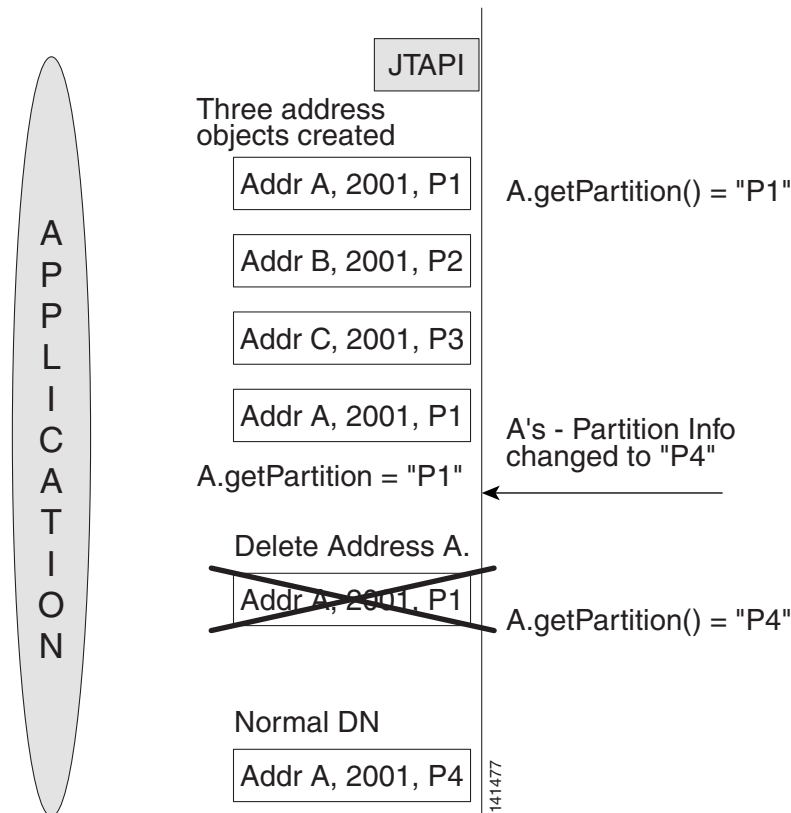
getParkingPartyPartition(), getParkedPartyPartition() and getParkPartyPartition() return the associated address objects as shown in the figure.

Change in Partition

Partition attribute is similar to the DN attribute of an address. Hence, whenever the partition attribute changes, the address object has to be destroyed and recreated. When the partition information of an address is changed, JTAPI will be restarted during which the current address objects will be deleted and new address objects will be created, reflecting the changed partition information.

Example 5 Change in Partition

Change in Partition Information



When the partition information of an address is changed, the address object will be destroyed and a new address object will be created.

The new address object will have the new partition information.

In the example given, Address A's partition string was changed to P4. Hence, the current address object of A will be deleted and a new address object will be created.

A query on the old address object using `A.getPartition()` will retrieve "P1", while the same query on the new object will return "P4".

When the address partition changes, applications should query the address objects to update their partition information.

Use Cases for JTAPI Partition Support

The common assumption for all of the following use cases is that CTI provides partition information for all the lines which the JTAPI opens in the response message and JTAPI stores the partition information for every address it maintains.

Table 1 Use Cases for JTAPI Partiton Support

S.No.	Pre-Condition	Use Case	Expected Behavior	Result.
1	A and B are two addresses in the same cluster with same DN and different partitions (P1, P2) and in same cluster. A calls B. CSS of A has B's partition in it first.	Calls between addresses with same DN in different device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B
2	A and B are two addresses with same DN in same device but different partitions (P1, P2) and in same cluster. A calls B. CSS of A has B's partition in it first.	Calls between addresses with same DN in same device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B.
3	A, B, and C are three different addresses with different DN. (P1, P2, P3). A park DN is configured with same DN as C and different partition (P4).	A calls B. B parks the call. C un parks the call from a park DN which is same as C's DN.	JTAPI should allow C to un park the call from park DN.	C is successfully able to un park the call from park DN.
4	A, B, and C are three different addresses having the same DN and different partitions (P1, P2, P3). A park DN is configured with same DN but belonging to a different partition (P4)	A calls B, B parks call at park DN. C un parks the call.	JTAPI should allow C to un park the call from park DN.	A is able to call B. B should be able to park the call at the park DN. C should be able to pick up the call.
5	A, B, and C are three different addresses with same DN and different partitions (P1, P2, and P3)	JTAPI calls getPartitionAddress(DN of A).	Three address objects are returned each corresponding to A, B and C.	JTAPI maintains the address objects based on partition info and DN.
6	A and B are addresses with same DN but belong to different partitions (P1, P2).	Application calls getPartition() on the address objects of A and B .		Partition strings of the addresses are returned correctly.

Table 1 Use Cases for JTAPI Partiton Support (continued)

7	A and B are two different addresses (different DNs) belonging to different partitions. A and B are in the control list of the same user. Provider open is completed and the lines are opened.	JTAPI supports old API to open lines when DN is different. There will be no change in behaviour.	Lines A and B will be opened, but since they have different DN, user need not specify the partition info. DN alone is sufficient to open the lines, but users can also give partition info and both modes of opening lines will be supported by JTAPI.	Address objects for A and B are created successfully.
8	A is an address in the control list of user and is opened by the user. From the CM admin page the partition information of A is changed. The device is restarted after this.	Partition information of a DN is changed. JTAPI should reflect new partition information.	JTAPI will delete the current address object and create a new address object for A when it comes in service again. By querying the partition info of this address object, user should get changed partition info.	New partition information is reflected in the new address object.

Hairpin Support

Use Cases for Hairpin Support

Table 2 Use Cases for Hairpin Support

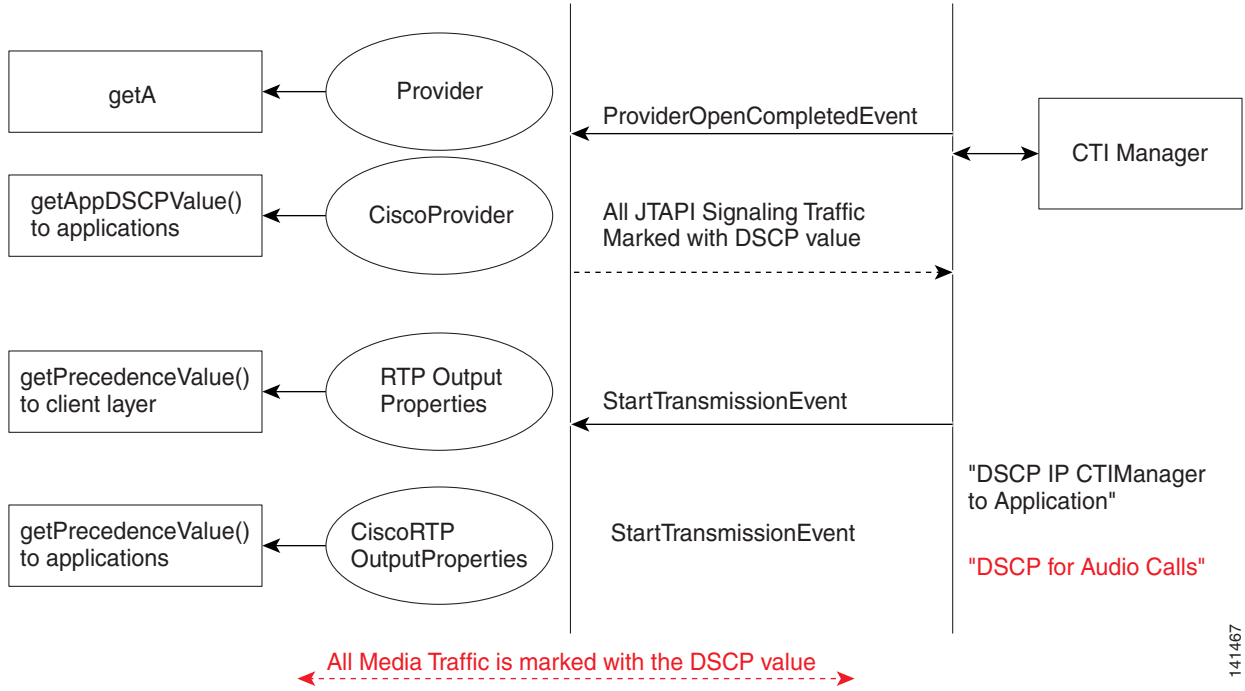
S.No.	Pre-Condition	Use Case	Expected Behaviour	Result.
1	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C. Gateway does not pass new party information to each other. There will be no transfer start and end events as transfer controller is not a controlled device.	A calls B via gateway. B transfers call to C via gateway. B completes the transfer and goes out of scenario. Now IP Phones A and C are connected	At A: It is connected to B. A's type is CiscoAddress.Internal B's type is CiscoAddress.External At C: It is connected to B. C's type is CiscoAddress.Internal B's type is CiscoAddress.External	A and C are connected.
2	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C. Gateway is able to pass new party information to each other.	A calls B via gateway. B transfers call to C via gateway. B completes the transfer and goes out of scenario. Now IP Phones A and C are connected.	At A: It is connected to C. A's type is CiscoAddress.Internal C's type is CiscoAddress.External At C: It is connected to A. C's type is CiscoAddress.Internal A's type is CiscoAddress.External	A and C are connected.

Table 2 Use Cases for Hairpin Support (continued)

3	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A and B.	A calls B. B does a conference call to C via gateway. B completes the conference and all A,B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
4	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A ,B and C.	A calls B. B does a conference call to C via gateway. B completes the conference and all A,B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
5	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway is able to pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A,B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
6	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway does not pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A,B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal D: CiscoAddress.External	A, B and C are in conference call.
7	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C.	A calls B via gateway. B redirects call to C. Now IP Phones A and C are connected.	At A: It is connected to C. A's type is CiscoAddress.Internal C's type is CiscoAddress.External At C: It is connected to A. C's type is CiscoAddress.Internal A's type is CiscoAddress.External	A and C are connected.

QoS Support


Figure A-1 Call Flow Diagram for QoS Support



141467

Use Cases for JTAPI QoS

For QoS to work in Windows, complete the following steps:

- Step 1** Start Registry Editor (Regedt32.exe).
 - Step 2** Go to the following key:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\
-  **Note** The registry key is one path.
- Step 3** On the Edit menu, click Add Value.
 - Step 4** Type DisableUserTOSSetting.
 - Step 5** Click REG_DWORD in the Data Type box.
 - Step 6** Click OK.
 - Step 7** Enter 0 in the prompt box.
 - Step 8** Quit the Registry Editor.
 - Step 9** Restart the computer.

For more information on this see <http://support.microsoft.com/default.aspx?scid=kb;en-us;248611>

Table 3 Use Cases for JTAPI QoS

Scenario	JTAPI Behavior
Application uses the JTAPI getPrecedenceValue() API to query for the new DSCP value, in CiscoRTPOutputStartedEvent.	JTAPI returns the DSCP value received from CTI in StartTransmissionEvent to the application.
Application uses the JTAPI getAppDSCPValue() API to query for the new DSCP value, when it gets ProviderInServiceEvent.	JTAPI returns the DSCP value received from CTI in ProviderOpenCompletedEvent to the application.

TLS Security

Message flow for updating certificate and establishing TLS certificate is illustrated in [Figure A-2](#) and [Figure A-3](#).

Figure A-2 CTI/API Security Approach

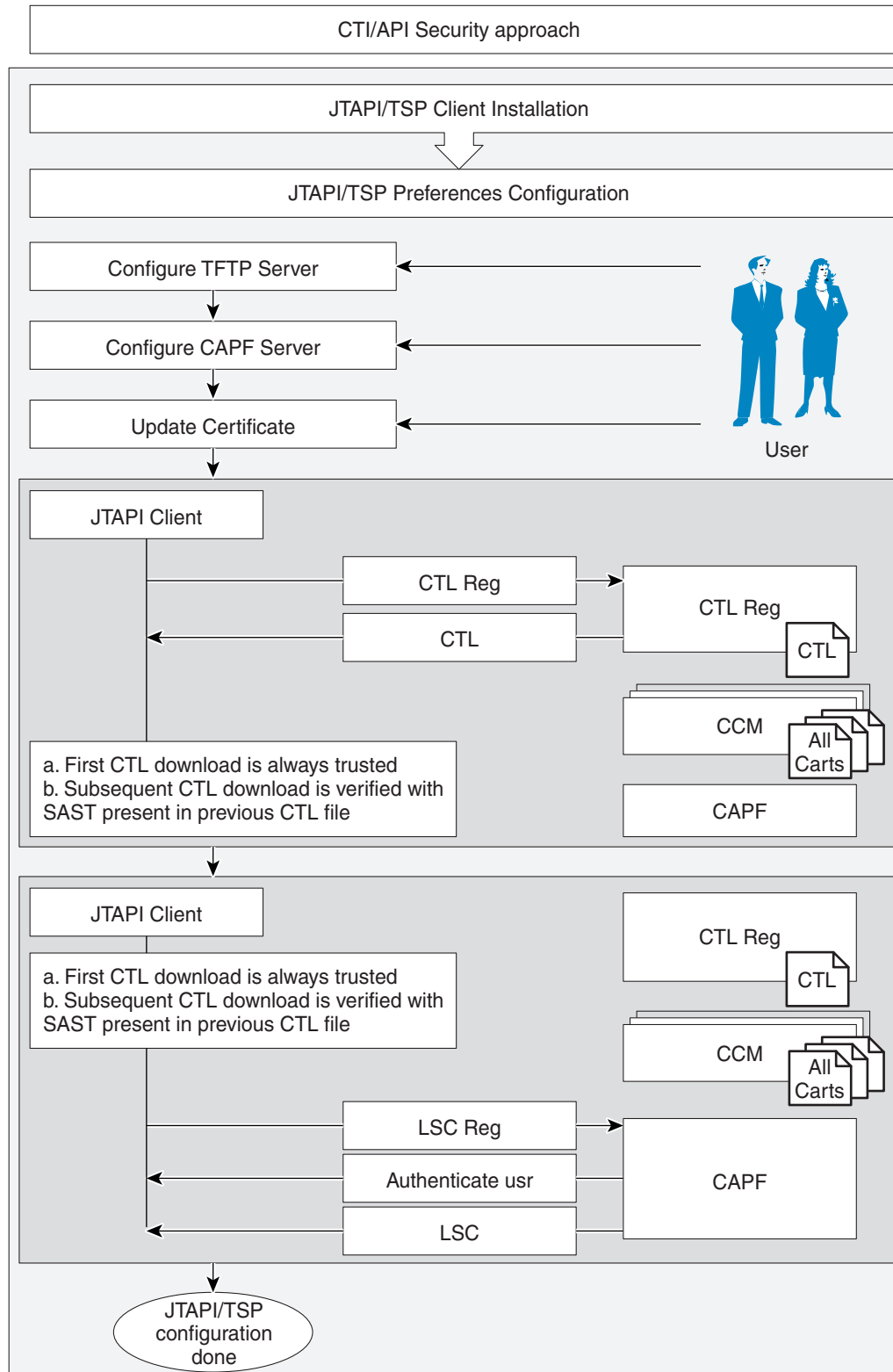
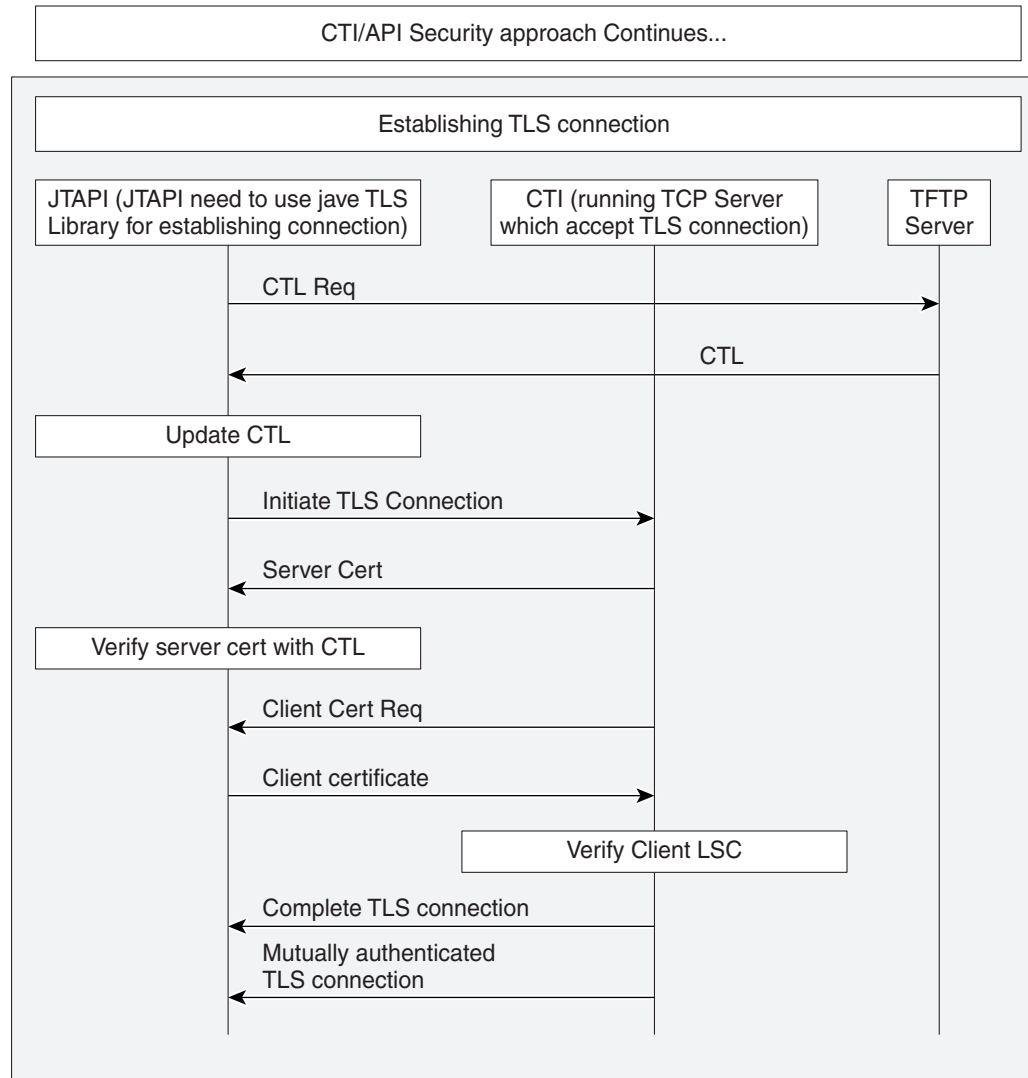


Figure A-3 CTI/API Security Approach (continued)



1411469

SRTP Key Material

If this feature is enabled, it is expected to degrade the performance of Cisco Unified CallManager JTAPI. Performance degradation is because of encrypted signaling between CTI and JTAPI and also because of encrypted media between end points.

Use Case 1

Action	Event
App adds CallObserver on an Address 1 and initiates a call to address2 and involves in secure media conversation.	CiscoRTPInputKeyEv CiscoRTPInputStartedEv CiscoRTPOutputKeyEv CiscoRTPOutputStartedEv
If user is authorized, then CiscoRTPInputKeyEv and CiscoRTPOutputKeyEv contain key material.	

Use Case 2

Action	Event
Application adds TerminalObserver by enabling snapshotEnabled filter. Device is already in a secure call and queries invokes CiscoTerminal.createSnapshot ()	CiscoTermSnapshotEv using which applications can query getCiscoMediaCallSecurity () to find out if a call is secured or not.

Use Case 3

Action	Response
Application does not have a TLS link and tries to register with secure media. CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm)	PrivilegeVoilationException is thrown to the application
Application has a secure media and registers CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm)	Request is successful

Device and Line Restriction

S.No	Scenario	Events
1	<p>Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list. T1 and A3 is added into the restricted list. Application opens the provider</p> <p>Application queries for isRestricted on T1,T2, T3</p> <p>Application queries for isRestricted on Address A1, A2, A3</p> <p>Application tries to addObserver and addCallObserver on T1,T2,T3, A1,A2,A3</p>	<p>CiscoTerminal.isRestricted() returns true for T1 and false for T2 and T3</p> <p>CiscoAddress.isRestricted() returns true for A1, A3, false for A2.</p> <p>CiscoAddress.getRestrictedAddrTerminals() on A1, A3 returns T1, T3 respectively, returns null for A2.</p> <p>addObserver and addCallObserver fails for T1, A1, A3. For T3 observer is added, but no events are received on A3. For A2, application will be able to add observers successfully and events will be received</p>
2	<p>Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.</p> <p>Application opens the provider and adds observer on all terminals and addresses.</p> <p>T1 and A2 are added to the restricted list.</p> <p>T1 and L2 are removed from restricted list</p>	<p>CiscoTermRestrictedEv for T1 CiscoAddrRestrictedEv for L1 CiscoAddrRestrictedEv for A2 sent to providerObserver.</p> <p>CiscoTermOutOfServiceEv for T1 CiscoAddrOutOfServiceEv for L1 CiscoAddrOutOfServiceEv for A2</p> <p>CiscoTermActivatedEv for T1 and CiscoAddrActivatedEv for A1 CiscoAddrActivatedEv for A2 sent to providerObserver.</p> <p>CiscoTermInServiceEv for T1and CiscoAddrInServiceEv for A1 CiscoAddrInServiceEv for A2 sent to terminal and address observers.</p>

3	<p>Application has Devices T1, T2, T3 whose lines are A1, A1, A2 in the control list. A1 is the shared line on T1 and T2</p> <p>Application opens provider and adds observer on all terminals/addresses</p> <p>T1 is added into the restricted list.</p> <p>T1 is removed from the restricted list</p>	<p>Application will see CiscoTermRestrictedEv for T1 and CiscoAddrRestrictedOnTerminalEv which contains getAddress is L1 and getTerminal as T1. Application will also see CiscoTermOutOfServiceEv for T1 and CiscoAddrOutOfService for A1/T1</p> <p>CiscoTermActivatedEv for T1 CiscoAddrActivatedEv for L1 CiscoTermInServiceEv for T1 CiscoAddrInServiceEv for A1/T1</p>
4	<p>Application has Devices T1, T2, T3 whose lines are A1, A1, A1 in the control list. A1 is the shared line on T1, T2 and T3</p> <p>Application opens the provider and adds observer on all terminals and addresses</p> <p>A1 on T1 is added to the restricted list</p> <p>A1 on T2 is added to the restricted list</p> <p>A1 on T3 is added to the restricted list</p> <p>A1 on T1 is removed from the restricted list</p> <p>A1 on T2 is removed from the restricted list</p> <p>A1 on T3 is removed from the restricted list</p>	<p>CiscoAddrRestrictedOnTerminalEv for A1/T1 CiscoAddrOutOfServiceEv for A1/T1</p> <p>CiscoAddrRestrictedOnTerminalEv for A1/T2 CiscoAddrOutOfServiceEv for A1/T2</p> <p>CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1/T3</p> <p>CiscoAddrActivatedOnTerminalEv for A1/T1 CiscoAddrInServiceEv for A1/T1</p> <p>CiscoAddrActivatedOnTerminalEv for A1/T2 CiscoAddrInServiceEv for A1/T2</p> <p>CiscoAddrActivatedEv for A1 CiscoAddrInServiceEv for A1/T3</p>
5	<p>Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.</p> <p>Application opens the provider and adds observer on all terminals and addresses. A1 is involved in a call with party X.</p> <p>A1 is added into the restricted list.</p>	<p>CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1</p> <p>ConnDisconnectedEv CallCtlConnDisconnectedEv TermConnDroppedEv CallCtlConnDroppedEv CallInvalidEv</p>

SIP Support

S.No	Scenario	Events
1	<p>External SIP phone (external@someserver.com) calls A, A is monitored by application.</p> <p>Assuming external sip phone uses uri and not DN.</p>	<p>Event delivered to call observer on A</p> <p>CallActiveEv ConnCreatedEv A ConnCreatedEv unknown</p> <p>.....</p> <p>getCurrentCallingPartyInfo().geUrlInfo().getUser() returns external.</p> <p>getCurrentCallingPartyInfo().geUrlInfo().getHost() returns someserver.com</p> <p>getCurrentCallingPartyInfo().geUrlInfo().getUrlType() returns SIP_URL_TYPE</p>
2	7970 runs SIP protocol with 2 max calls set. 3rd call comes in with GCID=GCID3	<p>GCID3 CallActiveEv GCID3 ConnCreatedEv A GCID3 ConnFailedEv A GCID3 callInvalidEv</p>
3	7960 running SIP is included in the control list. Applications add callobserver on the terminal	<p>Exception is thrown to addobserver exception. TerminalRestrictedEv will be delivered if the status changed.</p>

SIP REFER/REPLACE

REPLACES Message Flow

For the JTAPI events in the scenario described below, we have not shown Terminal events. It will be sent for all the observed Terminals as usual. Also events are shown with the assumption that only A, B, or C is observed; events would vary if combination of A, B, or C is observed.

SN	Scenario	Events at A	Events at B	Events at C
1.	<p>REPLACE with INVITE a confirmed Dialog:</p> <p>A (Dialog1) is in Call with B(Dialog2)(GC1). C sends INVITE with REPLACE Dialog2(GC2). After replace is completed, A(Dialog1) and C(Dialog3) are in the Call</p>	<p>GCID and CPIC with reason REPLACES, Cgpn=C, Cdpn=A, Ocdpn=A, Lrp=B</p> <p><u>JTAPI Events:</u></p> <p>CiscoCallChangedEv-(GC1 -GC2)</p> <p>ConnDisconnectedEv -B-GC1</p> <p>CallCtlConnDisconnectedEv-B-GC1</p> <p>ConnDisconnectedEv -A-GC1</p> <p>CallCtlConnDisconnectedEv-A-GC1 CallInvalid-GC1</p> <p>CallActive-CG2</p> <p>ConnCreatedEv -C-GC2</p> <p>ConnConnectedEv -C-GC2</p> <p>CallCtlConnEstablishedEv-C-CG2_</p> <p>ConnCreatedEv -A -GC2</p> <p>ConnConnectedEv -A-GC2</p> <p>CallCtlConnEstablishedEv-A-CG2</p> <p>Cause =CAUSE_NORMAL</p> <p>CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B</p>	<p>CSCE IDLE with reason REPLACES</p> <p><u>JTAPI Events:</u></p> <p>ConnDisconnectedEv -A -GC1</p> <p>CallCtlConnDisconnectedEv-A-GC1</p> <p>ConnDisconnectedEv -B-GC1</p> <p>CallCtlConnDisconnectedEv-B-GC1</p> <p>CallInvalidEv-GC1</p> <p>CAUSE_NORMAL</p> <p>Cause = CAUSE_NORMAL</p> <p>CiscoFeatureReason= REASON_REPLACES</p>	<p>NewCall/CSCE-Dialing/ CSCE-Connected with Cgpn=C, Cdpn=A, Ocdpn=B, Lrp=B</p> <p><u>JTAPI Events:</u></p> <p>CallActiveEv -GC2</p> <p>ConnCreatedEv -C -GC2</p> <p>ConnConnectedEv-C--GC2</p> <p>CallCtlConnEstablishedEv -C-GC2</p> <p>ConnCreatedEv -A-GC2</p> <p>ConnConnectedEv A--GC2</p> <p>CallCtlConnEstablishedEv -A--GC2</p> <p>Cause = CAUSE_NORMAL</p> <p>CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B</p>

<p>2.</p>	<p>REPLACE with INVITE an early Dialog: A (Dialog1) is in Call with B(Dialog2)(GC1) , B is ringing. C sends INVITE with REPLACE Dialog2(GC2). After replace completed, A(Dialog1) and C(Dialog3) in the Call</p>	<p>GCID and CPIC with reason REPLACES, Cgpn=C, Cdpn=A, Ocdpn=A, Lrp=B</p> <p><u>JTAPI Events</u> CiscoCallChangedEv-(GC1 -GC2) ConnDisconnectedEv -B-GC1 CallCtlConnDisconnectedEv-B-GC1 ConnDisconnectedEv -A-GC1 CallCtlConnDisconnectedEv-A-GC1 CallInvalid-GC1 CallActive-CG2 ConnCreatedEv -C-GC2 ConnConnectedEv -C-GC2 CallCtlConnEstablishedEv-C-CG2_ ConnCreatedEv -A -GC2 ConnConnectedEv -A-GC2 CallCtlConnEstablishedEv-A-CG2 Cause =CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u> Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B</p>	<p>CSCE-Idle with reason REPLACES</p> <p><u>JTAPI Events :</u> ConnDisconnectedEv -A -GC1 CallCtlConnDisconnectedEv-A-GC1 ConnDisconnectedEv -B-GC1 CallCtlConnDisconnectedEv-B-GC1 CallInvalidEv-GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p>	<p>NewCall/CSCE-Dialing/CS CE-Connected, with Cgpn=C, Ccdpn=A, Ocdpn=A, Lrp=B</p> <p><u>JTAPI Events:</u> CallActiveEv -GC2 ConnCreatedEv -C -GC2 ConnConnectedEv-C--GC2 CallCtlConnEstablishedEv-C-GC2 ConnCreatedEv -A-GC2 ConnConnectedEv A--GC2 CallCtlConnEstablishedEv-A--GC2 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u> Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B</p>
-----------	--	---	---	--

3.	<p>REPLACE with INVITE an early Dialog:</p> <p>A (Dialog1) is in Call with B(Dialog2) (GC1), B is ringing. C sends invite with replace Dialog-X (GC2)</p>			<p>NewCall/CSCE_Dialing/ reason REPLACES CSCE-Disconnected with reason REPLACES</p> <p><u>JTAPI Events:</u></p> <p>CallActiveEv –GC2 ConnCreatedEv –C–GC2 ConnConnectedEv –C–GC2 CallCtlConnEstablishedEv- C--GC2</p> <p>ConnFailedEv –C--GC2 ConnConnectedEv –C--GC2 CallCtlConnEstablishedEv- A--GC2</p> <p>Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=C, Called=, CurrentCalling=C, CurrentCalled=, LastRedirecting=</p>
4.	<p>REFER request with REPLACE Dialog:</p> <p>When REPLACE Dialog is in a Unified CM Cluster.</p> <p>A is in call with B(REFEREE) Dialog1, and Dialog2</p> <p>A is in Call with C(REFER TO TARGET) Dialog3 and Dialog4</p> <p>SIP-UA A send REFER B on Dialog1 to C with REPLACES Dialog3</p>	<p>TransferStartEv</p> <p>CSCE-Idle at Dialog1 with reason TRANSFER and at Dialog3 with reason TRANSFER</p> <p>TransferEndEv</p> <p><u>JTAPI Event:</u></p> <p>Regular TransferEvent</p>	<p>TransferStartEv</p> <p>CPIC with reason TRANSFER and Cgpn=B, Cdpn=C, Lrp=A OCdpn=C</p> <p>TransferEndEv</p> <p><u>JTAPI Event:</u></p> <p>Regular TransferEvents</p>	<p>TransferStartEv</p> <p>GCID with reason TRANSFER and Cgpn=B, Cdpn=C, Lrp=A OCdpn=C</p> <p>TransferEndEv</p> <p><u>JTAPI Event:</u></p> <p>Regular TransferEvents</p>

<p>5.</p>	<p>REFER request with REPLACE Dialog: When REPLACE Dialog is outside Unified CM Cluster SIP-UA A is in call with B, Dialog1 and Dialog2(GC1) SIP-UA A is in call with SIP-UA C Dialog3 SIP-UA A sends REFER B on Dialog1 to SIP-UA C with REPLACES Dialog3</p>	<p>No Events</p>	<p>CPIC with reason REFER and Cgpn=B, Cdpn=C, Lrp=A OCdpn=B <u>JTAPI Events:</u> ConnDisconnectedEv -A-GC1 CallCtlConnDisconnectedEv -A-GC1 ConnCreatedEv - C -GC1 ConnConnectedEv -C-GC1 CallCtlConnEstablishedEv -C-GC1 Cause = CAUSE_NORMAL CiscoFeatureReason=REASON_REFER <u>JTAPI CallInfo:</u> Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A</p>	<p>No Events</p>
-----------	--	------------------	---	------------------

6.	<p>REFER request with REPLACE Dialog:</p> <p>When A is outside a Unified CM Cluster</p> <p>SIP-UA A is in call with B, Dialog1 and Dialog2</p> <p>SIP-UA A is in call with C Dialog3 and Dialog4</p> <p>SIP-UA A sends REFER B on Dialog1 to C with REPLACES Dialog3</p>	No Events	<p>CPIC with reason REPLACES and Cgpn=B, Cdpn=C, Lrp=A, OCdpn=C</p> <p><u>JTAPI Events :</u></p> <p>ConnDisconnectedEv -A-GC1</p> <p>CallCtlConnDisconnectedEv-A-GC1</p> <p>ConnCreatedEv - C- GC1</p> <p>ConnConnectedEv -C -GC1</p> <p>CallCtlConnEstablishedEv -C-GC1</p> <p>Cause = CAUSE_NORMAL</p> <p>CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A</p>	<p>GCID with reason REPLACES and Cgpn=B, Cdpn=C, Lrp=A OCdpn=C</p> <p><u>JTAPI Events:</u></p> <p>CiscoCallChangedEv(GC2-GC1)_ConnDisconnectedEv -A-GC2</p> <p>CallCtlConnDisconnectedEv-A-GC2</p> <p>ConnDisconnectedEv -C-GC2</p> <p>CallCtlConnDisconnectedEv-C-GC2 CallInvalid-GC2</p> <p>CallActive-CG1</p> <p>ConnCreatedEv -B -GC1</p> <p>ConnConnectedEv -B-GC1</p> <p>CallCtlConnEstablishedEv-B-CG1</p> <p>ConnCreatedEv -C-GC1</p> <p>ConnConnectedEv -C-GC1</p> <p>CallCtlConnEstablishedEv-C-CG1_</p> <p>Cause = CAUSE_NORMAL</p> <p>CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A</p>
----	--	-----------	--	--

<p>7. REFER request with REPLACE Dialog: When REPLACE Dialog is in a Unified CM Cluster.</p> <p>A is in call with B(REFEREE) Dailog1, and Dialog2 (GC1)</p> <p>D is in Call with C(REFER TO TARGET) Dialog3 and Dialog4 (GC2)</p> <p>A sends REFER B on Dialog1 to C with REPLACES Dialog3</p> <p>B and C in final call.</p>	<p>CSCE-Idle at Dialog1 with reason REFER and at Dialog3 with reason REPLACES</p> <p><u>JTAPI Events:</u></p> <p>ConnDisconnectedEv -A -GC1 CallCtlConnDisconnectedEv-A-GC1 ConnDisconnectedEv -B-GC1 CallCtlConnDisconnectedEv-B-GC1 CallInvalidEv-GC1</p> <p>Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REFER</p> <p><u>Event at D:</u></p> <p>ConnDisconnectedEv -D -GC2 CallCtlConnDisconnectedEv-D-GC2</p> <p>ConnDisconnectedEv -C-GC2 CallCtlConnDisconnectedEv-C-GC2 CallInvalidEv-GC2</p> <p>Cause = CAUSE_NORMAL CiscoFeatureReason=REASON_REPLACES</p>	<p>CPIC with reason REPLACES and Cgpn=B, Cdpn=C, Lrp=D OCdpn=C</p> <p><u>JTAPI Events:</u></p> <p>ConnDisconnectedEv -A-GC1 CallCtlConnDisconnectedEv-A-GC1 ConnCreatedEv -C-GC1 ConnConnectedEv -C -GC1 CallCtlConnEstablishedEv-C-GC1</p> <p>Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=D</p>	<p>GCID with reason REPLACES and Cgpn=B, Cdpn=C, Lrp=D, OCdpn=C</p> <p><u>JTAPI Events:</u></p> <p>CiscoCallChangedEv(GC2-GC1)_ConnDisconnectedEv -D CallCtlConnDisconnectedEv-D ConnDisconnectedEv -C CallCtlConnDisconnectedEv-C CallInvalid-GC2 CallActive-CG1 ConnCreatedEv -C-GC1 ConnConnectedEv -C-GC1 CallCtlConnEstablishedEv-C-CG1_</p> <p>ConnCreatedEv -B -GC1 ConnConnectedEv -B-GC1 CallCtlConnEstablishedEv-B-CG2</p> <p>Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES</p> <p><u>JTAPI CallInfo:</u></p> <p>Calling=C, Called=C, CurrentCalling=B, CurrentCalled=C, LastRedirecting=D</p>
---	--	--	---

REFER Message Flow

The following section describes the scenarios that might be encountered during a SIP REFER. There are two categories of REFER scenarios: IN-Dialog and OutOfDialog.

Scenarios for IN-Dialog REFER

There are 11 scenarios (A through K) described in the sections that follow for IN-Dialog REFERs.

Scenario A

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C is Ringing.

JTAPI moves A's Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection into the "UNKNOWN" state.

CAUSE_CODE provided will be CAUSE_NORMAL, new API provides REASON_REFER.

For C a new Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection would be created.

CallInfo at B and C would be as follows:

At B: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

At C: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

Scenario B

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C Answers the Call.

JTAPI will Disconnect/Drop A's Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API would provide REASON_REFER.

For C Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will move to the Connected/Established/Active/Talking state.

CallInfo at B and C will be as follows

At B: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

At C: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

Scenario C

A(SIP UA inside cluster) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C is ringing but C did not answer the call and has no forward configured. Refer fails, the original call between A and B is restored.

JTAPI will Disconnect/Drop the Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for C. CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER and move A's Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection from the "Unknown" state to the Connected/Established/Active/Talking state.

CallInfo at A and B will be as follows

At A: Cgpn=A, Cdpn=B, Lrp= OCdpn=B

At B: Cgpn=A, Cdpn=B, Lrp= OCdpn=B

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()= NULL

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()=NULL

Scenario D

A(SIP UA outside cluster) is in call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C is ringing.

JTAPI will create Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for C and will drop A's Connection/CallControlConnection on getting CPIC at B, CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER.

CallInfo at B and C will be as follows:

At B: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

At C: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

Scenario E

A(SIP UA outside cluster) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C is ringing but C did not answer the call and has no forward configured. Refer fails, the original Call between A and B is restored.

JTAPI will create Connection/CallControlConnection for A again and drops Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for C. CAUSE_CODE provided will be CAUSE_NORMAL and new API will provide REASON_REFER.

CallInfo at A and B will be as follows

At A: Cgpn=A, Cdpn=B, Lrp= OCdpn=B

At B: Cgpn=A, Cdpn=B, Lrp= OCdpn=B

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()=NULL

JTAPI Application observing C will see:

```
getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty()=A
getCurrentCalledParty()=B
getLastRedirecting()=NULL
```

Scenario F

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C answers the call.

JTAPI moves Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for C to the Connected/Established/Active/Talking state. CAUSE_CODE provided is CAUSE_NORMAL and the new API will provide REASON_REFER.

CallInfo at B and C will be as follows

```
At B: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C
At C: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C
```

JTAPI Application observing B will see:

```
getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty()=B
getCurrentCalledParty()=C
getLastRedirecting()=A
```

JTAPI Application observing C will see:

```
getCallingParty() = B
getCalledParty() = C
getCurrentCallingParty()=B
getCurrentCalledParty()=C
getLastRedirecting()=A
```

Scenario G

A(SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C forwardAll to D, D is ringing.

JTAPI creates Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for D. CAUSE_CODE provided will be CAUSE_REDIRECT and the reason received from CTI would be ForwardAll.

CallInfo at B and D will be as follows

```
At B: Cgpn=B, Cdpn=D, Lrp=C OCdpn=C
At D: Cgpn=B, Cdpn=D, Lrp=C OCdpn=C
```

JTAPI Application observing B will see:

```
getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty()=B
getCurrentCalledParty()=D
getLastRedirecting()=C
```

JTAPI Application observing D will see:

```
getCallingParty() = B
getCalledParty() = D
getCurrentCallingParty()=B
getCurrentCalledParty()=D
getLastRedirecting()=C
```

Scenario H

A (SIP UA in cluster/in control) is in a call with B.

A(referrer) REFERS B(Referee) to C(Refer to target), C Redirect to D, D is ringing.

JTAPI creates Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection for D. CAUSE_CODE provided will be CAUSE_REDIRECT and the reason received from CTI in NewCallEvent at D will be Redirect.

Callinfo when Call is offered at C:

```
At B: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C
At C: Cgpn=B, Cdpn=C, Lrp=A OCdpn=C
```

CallInfo in final Call :

```
At B: Cgpn=B, Cdpn=D, Lrp=C OCdpn=C
At D: Cgpn=B, Cdpn=D, Lrp=C OCdpn=C
```

JTAPI Application observing B will see in final Call:

```
getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty()=B
getCurrentCalledParty()=D
getLastRedirecting()=C
```

JTAPI Application observing D will see:

```
getCallingParty() = B
getCalledParty() = D
getCurrentCallingParty()=B
getCurrentCalledParty()=D
getLastRedirecting()=C
```

Scenario I

A(SIP UA in cluster/in control) is in a call with B.
B consult transfer to D, A(Referrer) REFERS B(Referee) to C(Refer to target), C is ringing,
B completes the transfer. Attempt to transfer will fail while C is ringing.

Scenario J

A(SIP UA in cluster/in control) is in a call with B.
B consult transfer to D, A(Referrer) REFERS B(Referee) to C(Refer to target), C answers the call.
Refer will be successful. B completes the transfer, transfer will be successful, C and D will be in call.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/
CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new
API will provide REASON_REFER.

For C, Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will
move to Connected/Established/Active/Talking state.

CallInfo at D and C would be as follows

At D: Cgpn= C, Cdpn=D, Lrp=B OCdpn=D

At C: Cgpn=C, Cdpn=D, Lrp=B OCdpn=D

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty()=C

getCurrentCalledParty()=D

getLastRedirecting()=B

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=C

getCurrentCalledParty()=D

getLastRedirecting()=B

Scenario K

B is in a call with D, B consults to A(SIP UA in cluster/in control).
A(Referrer) REFERS B(Referee) to C(Refer to target), C is ringing, B completes the transfer.
REFER would fail. Call at A will be dropped, transfer is successful, D is getting RingBack, C is ringing.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/
CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API
would provide REASON_REFER, Application will not know if REFER failed.

For C, Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection will
move to Alerting/Alerting/Ringing/Ringing state.

CallInfo at D and C would be as follows:

At D: Cgpn= D, Cdpn=C, Lrp=B OCdpn=C

At C: Cgpn=D, Cdpn=C, Lrp=B OCdpn=C

JTAPI Application observing D will see:

```
getCallingParty() = B
getCalledParty() = D
getCurrentCallingParty()=D
getCurrentCalledParty()=C
getLastRedirecting()=B
```

JTAPI Application observing C will see:

```
getCallingParty() = B
getCalledParty() = C
getCurrentCallingParty()=D
getCurrentCalledParty()=C
getLastRedirecting()=B
```

For OutOfDialog Refer

SIP-UA A REFERS B(Referee) to C (Refer To Target)

B gets newcall with Cgpn=A, Cdpn=B, Lrp=, OCdpn=B.

JTAPI Application will get CallActive, Connection, CallCtlConnection, TerminalConnecton and CallCtlTerminalConnection created for B with CAUSE_NORMAL, and the new API will return REASON_REFER.

B's Connection/CallCtlConnection, TerminalConnection/CallCtlTerminalConnection will go into the Connected/Established/Active/Talking state. JTAPI creates Connection and CallCtlConnection for A in "UNKNOWN" state based on FarEndPointType_ServerCall provided by CTI/CP.

B answers the call and is connected to A (at this point no RTPEvent will be sent).

B get CallPartyInfoChangedEv with Cgpn=B, Cdpn=C, Lrp=A, OCdpn=C, Reason=REFER.

C get NewCall offering with Cgpn=B, Cdpn=C, Lrp=A, OCdpn=C, Reason=REFER.

JTAPI Application will get Connection, CallControlConnection, TerminalConnecton and CallCtlTerminalConnection created for B with CAUSE_NORMAL, and the new API will return REASON_REFER.

C Accepts/Answers the call, B is connected to C (now Application receives RTP events).

C's Connection/CallCtlConnection, TerminalConnection/CallCtlTerminalConnection will go into the Connected/Established/Active/Talking state.

JTAPI Application observing B will see:

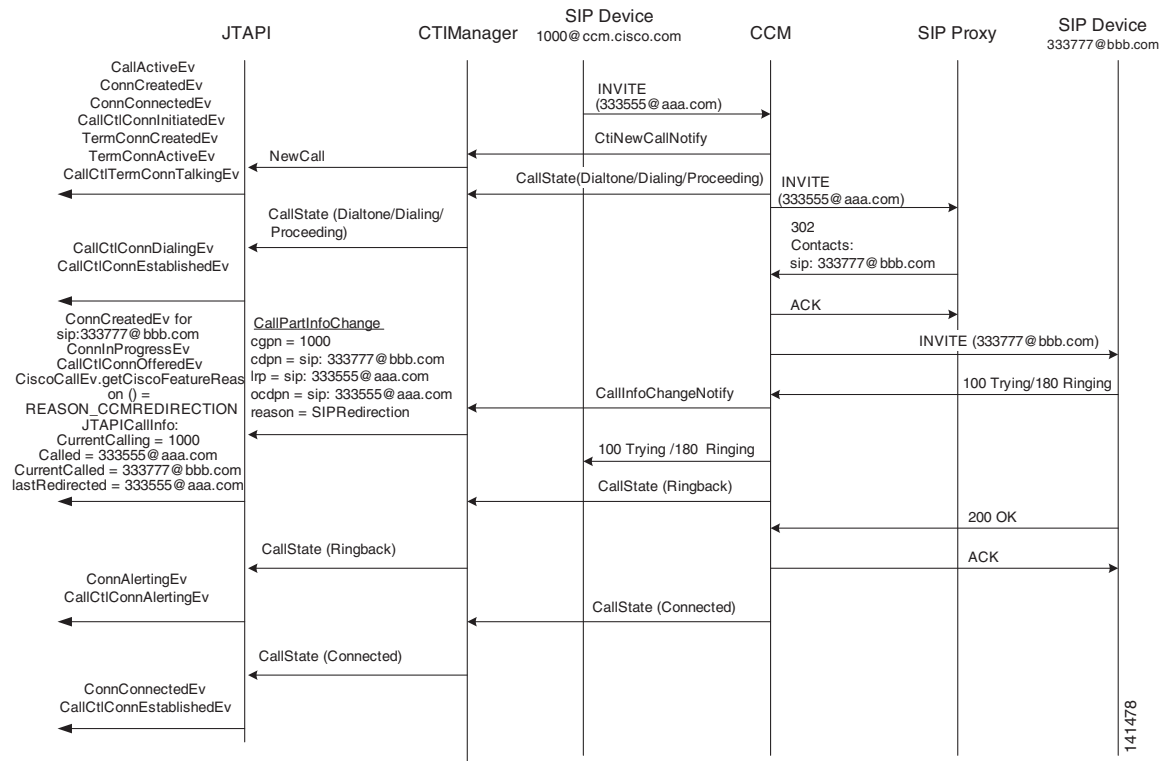
```
getCallingParty() = A
getCalledParty() = B
getCurrentCallingParty()=B
getCurrentCalledParty()=C
getLastRedirecting()=A
```

JTAPI Application observing C will see:

- getCallingParty() = B
- getCalledParty() = C
- getCurrentCallingParty()=B
- getCurrentCalledParty()=C
- getLastRedirecting()=A

SIP 3XX Redirection

3XX Redirection – 302 Moved Temporarily



JTAPI application monitors 1000@ccm.cisco.com

Unified CM user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and Connection and CallCtlConnection events for 1000

JTAPI reports CallCtlConnEstablishedEv

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302, the Unified CM initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com.

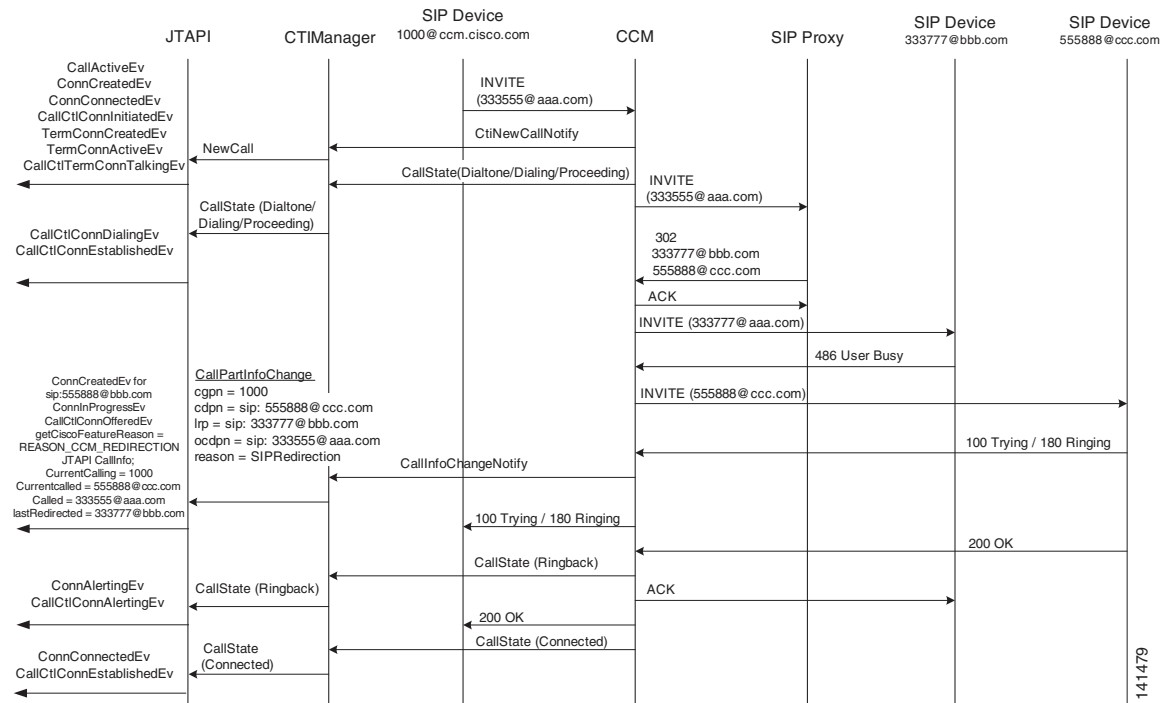
CallPartyInfoChange event is reported to application based on the SIPAlertInd from a Unified CM, if the called party information is changed.

JTAPI reports connection created events for 333777@bbb.com

CTI reports CtiCallStateNotify (Ringback) and CtiCallStateNotify (Connected).

JTAPI reports ConnAlertingEv and ConnEstablishedEv for far end.

3XX Redirection – Contact Busy



JTAPI CTI application monitors 1000@ccm.cisco.com

Unified CM user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and Connection and CallCtiConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtiConnEstablishedEv

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Unified CM initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com.

A 486 user busy response is reported by 333777@bbb.com. Based on this response the Unified CM initiates a call to 555888@cisco.com.

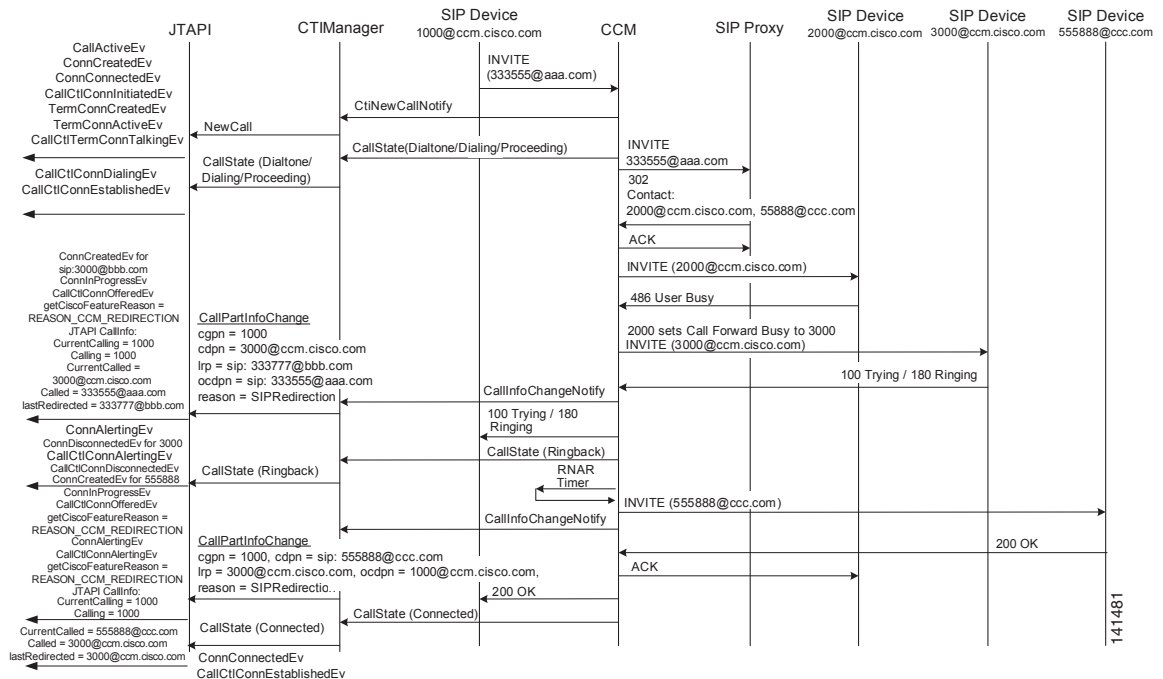
CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Unified CM if the called party information is changed.

JTAPI reports connection created event for 555888@cisco.com.

CTI also reports CtiCallStateNotify (Ringback) and CtiCallStateNotify (Connected).

JTAPI reports CallCtiConnAlertingEv and CallCtiConnEstablishedEv for the new party

3XX Redirection – Contact Within Unified CM Cluster Configured with Call Forward



JTAPI application monitors 1000@ccm.cisco.com

Unified CM user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtiConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Unified CM initiates a call to the first contact in the Target list based on the q value to 2000@ccm.cisco.com.

A 486 user busy response is reported by 2000@ccm.cisco.com. 2000 has Call Forward busy configured so the Unified CM initiates a call to 3000@ccm.cisco.com. The Unified CM also starts the RNAR timer.

CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Unified CM if the called party information is changed.

JTAPI reports connection created event for 3000

3000 does not answer and RNAR timer expires and based on this expiration the Unified CM initiates a call to 555888@cisco.com.

CallPartyInfoChange event is reported to application based on the SIPAlertInd/CcNotifyReq from the Unified CM if the called party information is changed.

JTAPI destroys connection for 3000 and creates connection for 555888

CTI also reports CtiCallStateNotify (Connected).

JTAPI reports CallCtiConnEstablishedEv for 555888

3XX Redirection – Non-Available Target Member

JTAPI application monitors 1000@ccm.cisco.com

Unified CM user1000 initiates a call to 333555@aaa.com

CTI reports NewCallNotify and CtiCallStateNotify (Dialtone/Dialing) based on INVITE.

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. 302 contains target list of 1212@ccm.cisco.com and 2000@ccm.cisco.com. 1212@ccm.cisco.com is an invalid DN. The Unified CM tries to contact 1212@ccm.cisco.com first, but gets an invalid DN and so attempts to place the call to 2000@ccm.cisco.com.

CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Unified CM if the called party information is changed.

JTAPI reports connection created event for 2000

CTI also reports CtiCallStateNotify (Ringback/Connected).

JTAPI reports CallCtlConnAlertingEv and CallCtlConnEstablishedEv for 2000.

Unicode Support

Use Cases for Unicode Display Name

Scenario	Events Delivered to JTAPI Applications
A line is configured on IP phone A with no ASCII name and a Unicode name in Japanese. IP phone B is configured with ASCII name and no Unicode name. A calls B. Only B is observed.	Call info should contain: <pre> getCurrentCalledPartyDisplayName=asciiNameB getCurrentCalledPartyUnicodeDisplayName=null getCurrentCallingPartyDisplayName=null getCurrentCallingPartyUnicodeDisplayName=japaneseNameA </pre>
A, B and C are in conference.	DisplayName does not apply. Applications should consider “conference” as the called party.
Shared lines – A and B are shared lines with different locales. A calls C. C is unobserved.	Calling party Unicode display name can change between A and B.

Use Cases to Get Locale and UniCodeCapabilities of Terminal

Scenario	Events delivered to JTAPI applications
A line is configured on IP phone A with no ASCII name and Unicode name in Japanese.	
Application adds TerminalObserver on the Device.	CiscoTerminalInServiceEv contains getLocale = JAPANESE getSupportedEncoding= UCS2UNICODE_ENCODING
Application queries the following using CiscoTerminal.	CiscoTerminal.getLocale = JAPANESE CiscoTerminal.getSupportedEncoding= UCS2UNICODE_ENCODING

Backward Compatibility Enhancements

This feature is not expected to change the performance or scalability of Cisco Unified CallManager JTAPI. There is no change in the number of events between JTAPI and CTI. For features involving GCID changes this feature introduces one extra event which should not cause any performance issues.

In all cases events listed below are delivered to call observers when only one party is in control list. TERMA indicates terminal of A.

Scenario A

A calls B, B transfers the call to C. GC1 is the call between A and B, GC2 is the consult call between B and C. Similar events are delivered for Conference and other features.

Action	Events
B completes the transfer. Events to call observer on C	GC2 CiscoTransferStartEv Cause: CAUSE_NORMAL Reason=REASON_TRANSFER CallActiveEv GC1 Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER ConnCreatedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER CiscoCallChangedEv SurvingCall=GC1, original call=GC2 CiscoCause: NORMAL Reason: REASON_TRANSFER

Action	Events
Events delivered to CallObserver of B (transfer controller)	<p>ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>TermConnCreatedEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>TermConnActiveEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>CallCtlTermConnTalkingEv TERM C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>ConnConnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER</p> <p>GC2: ConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_NORMAL</p> <p>GC2: ConnDisconnectedEv C REASON=REASON_TRANSFER Cause: CAUSE_NORMAL</p> <p>GC2: TermConnDropped TERMB REASON=REASON_TRANSFER Cause: CAUSE_NORMAL</p> <p>GC2: CalInvalid REASON=REASON_TRANSFER Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlTermConnHeldEv TERMB REASON=REASON_TRANSFER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL</p> <p>GC2: ConsultCallActive REASON=NORMAL Cause: CAUSE_NEW_CALL</p> <p>GC2: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL</p> <p>GC2: ConnConnectedEv B REASON=NORMAL Cause: CAUSE_NORMAL</p> <p>GC1: ConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN</p>

Action	Events
Events delivered to CallObserver of B (transfer controller) (continued)	<p>GC1: CallCtlConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER</p> <p>GC1: TermConnDroppedEv TERMB REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN</p> <p>GC1: CallCtlTermConnDroppedEv TERMB REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER</p> <p>GC1: ConnDisconnectedEv A REASON=REASON_TRANSFER</p> <p>GC1: CallCtlConnDisconnectedEv A REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER</p> <p>GC1: CallInvalidEv REASON=REASON_TRANSFER</p> <p>GC2: ConnDisconnectedEv C REASON=REASON_TRANSFER</p> <p>GC2: CallCtlConnDisconnectedEv C REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER</p> <p>GC2: TermConnDroppedEv TERMB REASON=REASON_TRANSFER</p> <p>GC2: CallCtlTermConnDroppedEv TERMB REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER</p> <p>GC2: ConnDisconnectedEv B REASON=REASON_TRANSFER</p> <p>GC2: CallCtlConnDisconnectedEv B REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER</p> <p>GC2: CallInvalidEv REASON=REASON_TRANSFER</p> <p>GC2: CallObservationEndedEv REASON=NORMAL Cause: CAUSE_NORMAL</p> <p>GC1 CiscoTransferEndEv REASON=REASON_TRANSFER Cause: CAUSE_NORMAL</p> <p>GC2 CallObservationEndedEv REASON=NORMAL Cause: CAUSE_NORMAL</p>

Scenario B

A calls B, call=GC1. B parks the call at 99999. C unparks the call using call GC2.

Action	Events
Events delivered to call observer on A when call is parked.	GC1: ConnDisconnectedEv B REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON=REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnCreatedEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: ConnInProgressEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnQueuedEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK
When call is unparked using GC2	GC2: CiscoCallChangedEv Surviving= GC2 origcall= GC1 address= A REASON=REASON_UNPARK CallActiveEv REASON=REASON_UNPARK Cause: CAUSE_NEW_CALL GC2: ConnCreatedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: TermConnCreatedEv TERMA REASON=REASON_UNPARK GC2: TermConnActiveEv TERMA REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TERMA REASON=REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK

GC1: ConnDisconnectedEv 9999
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL

GC1: CallCtlConnDisconnectedEv 9995
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL
 CallControlCause: CAUSE_PARK

GC1: TermConnDroppedEv TERMA
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL

GC1: CallCtlTermConnDroppedEv TERMA
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL
 CallControlCause: CAUSE_PARK

GC1: ConnDisconnectedEv A
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL

GC1: CallCtlConnDisconnectedEv A
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL
 CallControlCause: CAUSE_PARK

GC1: CallInvalidEv
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL

GC1: CallObservationEndedEv
 REASON=NORMAL
 Cause: CAUSE_NORMAL

GC2: ConnCreatedEv C
 REASON=REASON_UNPARK
 Cause: CAUSE_NORMAL

GC2: ConnConnectedEv C
 REASON=UNPARK
 Cause: CAUSE_NORMAL

GC2: CallCtlConnEstablishedEv C
 REASON=UNPARK
 Cause: CAUSE_NORMAL
 CallControlCause: CAUSE_PARK

Scenario C

A calls B, B has forward no answer to C. B does not answer and call is offered to C.

Action	Events
Events delivered to call observer on A.	GC1: CallActiveEv REASON=NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnInitiatedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMA REASON=NORMAL GC1: TermConnActiveEv TERMA REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnTalkingEv TERMA REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnDialingEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Events delivered to call observer on A. (continued)	<p>GC1: ConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL</p> <p>GC1: ConnCreatedEv C REASON=REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL</p> <p>GC1: ConnInProgressEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlConnOfferedEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED</p> <p>GC1: ConnAlertingEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlConnAlertingEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL</p> <p>GC1: ConnDisconnectedEv B REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlConnDisconnectedEv B REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED</p> <p>GC1: ConnConnectedEv C REASON=NORMAL Cause: CAUSE_NORMAL</p> <p>GC1: CallCtlConnEstablishedEv C REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL</p>
---	---

Scenario D

A call B, B redirects the call to C.

Action	Events
Events delivered to call observer on B.	GC1: CallActiveEv REASON=NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMB REASON=NORMAL Cause: Other: 0 GC1: TermConnRingingEv TERMB REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEvImpl TERMB REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Events delivered to call observer on B. (continued)	GC1: ConnDisconnectedEv A REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: TermConnDroppedEv TERMB REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMB REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnDisconnectedEv B REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: CallInvalidEv REASON=REDIRECT Cause: CAUSE_NORMAL
---	---

Half Duplex Media Support Message Flow

RTP Event at A and B.

Action	RTP Events	Check Interface
A calls B , B answers the call.	A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false
B puts Call on hold	A – CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv B – CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv A-CiscoRTPInputStartedEv	Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns True
B Retrieves the Call	A- CiscoRTPInputStoppedEv A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() returns True Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false Ev.isHalfDuplex() returns false