Message Sequence Charts

This appendix contains message sequence charts illustrating the message flows for several scenarios.

Agent Greeting

The basic Agent Greeting use cases assume a common setup.

In the real-world scenario, an external customer calls a number and is routed through an IVR until the call is eventually offered to an agent.

IP Phones:

- Customer (1000)
- Agents (2000, 2001, 2002)
- IVRs (5000, 5001)

Scenario One

Agent Greeting Start Success

Action	Events	Call information / Notes
1. Customer dials the agent.	GC1 - CallActiveEvent GC1 - ConnCreatedEvent (1000) GC1 - ConnConnectedEvent (1000) GC1 - CallCtlConnInitiatedEv (1000) GC1 - TermConnCreatedEvent (Term of 1000) GC1 - TermConnActiveEvent (Term of 1000) GC1 - CallCtlTermConnTalkingEv (Term of 1000) GC1 - CallCtlConnDialingEv (1000) GC1 - CallCtlConnEstablishedEv (1000) GC1 - ConnCreatedEvent (2000) GC1 - ConnInprogressEvent (2000) GC1 - CallCtlConnOfferedEv (2000) GC1 - ConnAlertingEvent (2000) GC1 - CallCtlConnAlertingEv (2000) GC1 - TermConnCreatedEvent (Term of 2000) GC1 - TermConnRingingEvent (Term of 2000) GC1 - CallCtlTermConnRingingEv (Term of 2000) GC1 - ConnConnectedEvent (2000) GC1 - CallCtlConnEstablishedEv (2000) GC1 - TermConnActiveEvent (Term of 2000) GC1 - CallCtlTermConnTalkingEv (Term of 2000)	This is a basic call. Calling = 1000 (Customer) Called = 2000 (Agent)

Action	Events	Call information / Notes
2. Application gets the TerminalConnection for 2000 on GC1 and invokes addMediaStream("5000", "2000")	GC2 - CallActiveEvent GC2 - ConnCreatedEvent (5000) GC2 - ConnInprogressEvent (5000) GC2 - CallCtlConnOfferedEv (5000) GC2 - ConnAlertingEvent (5000) GC2 - CallCtlConnAlertingEv (5000) GC2 - TermConnCreatedEvent (Term of 5000) GC2 - TermConnRingingEvent (Term of 5000) GC2 - CallCtlTermConnRingingEv (Term of 5000) GC2 - ConnConnectedEvent (5000) GC2 - CallCtlConnEstablishedEv (5000) GC2 - TermConnActiveEvent (Term of 5000) GC2 - CallCtlTermConnTalkingEv (Term of 5000) GC1 - CiscoMediaStreamStartedEv (2000)	This is a server call. Calling = 2000 (Agent) Called = 5000 (IVR) The Calling Party number is as specified in the addMediaStream() method ("2000" in this case), and is available immediately from the CallActiveEvent. **Note** No connection for 2000 is created, as 2000 is "spoofed". Agent Greeting is complete.
3. Application disconnects IVR, or tester manually hangs up the IVR device.	GC2 - CallCtlTermConnDroppedEv (Term of 5000) GC2 - ConnDisconnectedEvent (5000) GC2 - CallCtlConnDisconnectedEv (5000) GC2 - CallInvalidEvent (5000) GC2 - CallObservationEndedEv GC1 - CiscoMediaStreamEndedEv (2000)	BIB call is cleaned up. Ev.isSuccessful() = true. The call continues as normal.
4. Agent finishes the conversation and ends the call	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000) GC1 - CallCtlConnDisconnectedEv (2000) GC1 - TermConnDroppedEv (Term of 1000) GC1 - CallCtlTermConnDroppedEv (Term of 1000) GC1 - ConnDisconnectedEvent (1000) GC1 - CallCtlConnDisconnectedEv (1000) GC1 - CallInvalidEvent GC1 - CallObservationEndedEv	Primary call is cleaned up.

Scenario Two

Agent Greeting Stop Success

Agent	Events	Call information / Notes
1. Customer calls the agent and the agent answers. Application invokes addMediaStream().	GC1 - CiscoMediaStreamStartedEv (2000)	Ev.getIVRCall() = Call for CG2.
2. While the greeting is played, the application invokes removeMediaStream().	GC2 - CallCtlTermConnDroppedEv (Term of 5000) GC2 - ConnDisconnectedEvent (5000) GC2 - CallCtlConnDisconnectedEv (5000) GC2 - CallInvalidEvent (5000) GC2 - CallObservationEndedEv GC1 - CiscoMediaStreamEndedEv (2000)	The Agent Greeting is cut short. The BIB call is cleaned up. Ev.isSuccessful() = true. The call continues as normal.
3. The agent finishes the conversation and ends the call.	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000) GC1 - CallCtlConnDisconnectedEv (2000) GC1 - TermConnDroppedEv (Term of 1000) GC1 - CallCtlTermConnDroppedEv (Term of 1000) GC1 - ConnDisconnectedEvent (1000) GC1 - CallCtlConnDisconnectedEv (1000) GC1 - CallInvalidEvent GC1 - CallObservationEndedEv	The primary call is cleaned up.

Scenario Three

Agent Greeting Start Failure: Resource Unavailable

Agent	Event	Call information / Notes
1. Customer dials the Agent	GC1 - CallActiveEvent GC1 - ConnCreatedEvent (1000) GC1 - ConnConnectedEvent (1000) GC1 - CallCtlConnInitiatedEv (1000) GC1 - TermConnCreatedEvent (Term of 1000) GC1 - TermConnActiveEvent (Term of 1000) GC1 - CallCtlTermConnTalkingEv (Term of 1000) GC1 - CallCtlConnDialingEv (1000) GC1 - CallCtlConnEstablishedEv (1000) GC1 - ConnCreatedEvent (2000) GC1 - ConnInprogressEvent (2000) GC1 - CallCtlConnOfferedEv (2000) GC1 - ConnAlertingEvent (2000) GC1 - CallCtlConnAlertingEv (2000) GC1 - TermConnCreatedEvent (Term of 2000) GC1 - TermConnRingingEvent (Term of 2000) GC1 - CallCtlTermConnRingingEv (Term of 2000) GC1 - ConnConnectedEvent (2000) GC1 - CallCtlConnEstablishedEv (2000) GC1 - TermConnActiveEvent (Term of 2000) GC1 - CallCtlTermConnTalkingEv (Term of 2000)	This is a basic call Calling = 1000 (Customer) Called = 2000 (Agent)
2. The application gets the TerminalConnection for 2000 on GC1and invokes addMediaStream("5000", "2000").		No BIB call is created. JTAPI throws a ResourceUnavailableException with text as "Unable to allocate built in bridge resource". The call continues as normal.
3. The agent finishes the conversation and ends the call.	GC1 - TermConnDroppedEv (Term of 2000) GC1 - CallCtlTermConnDroppedEv (Term of 2000) GC1 - ConnDisconnectedEvent (2000)	The primary call is cleaned up.

API for Exposing Built-in-Bridge Status

Phone TermA, CTI port TermB, and RoutePoint TermC are in application’s control list.

Use Case One

BIB is disabled on service parameters and device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	False	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Two

BIB is disabled on service parameters page and enabled on device page of TermA..

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	True	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Three

BIB is enabled on service parameters page and disabled on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	False	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Four

BIB is enabled on service parameters page and set to default on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	True	
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Use Case Five

Phone TermA is not registered. BIB is enabled on device page of TermA.

Action	Result	Call information
TermA.isBuiltInBridgeEnabled()	InvalidStateException	
Add observers and register TermB and TermC.		
TermB.isBuiltInBridgeEnabled()	MethodNotSupportedException	
TermC.isBuiltInBridgeEnabled()	MethodNotSupportedException	

Backward Compatibility Enhancements

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

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

Scenario One

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

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

Action	Events
Events delivered to CallObserver of B (transfer controller)	

Action	Events
	ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER TermConnCreatedEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER TermConnActiveEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER CallCtlTermConnTalkingEv TERM C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER ConnConnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason = REASON_TRANSFER GC2: ConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: ConnDisconnectedEv C REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: TermConnDropped TERMB REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2: CalInvalid REASON = REASON_TRANSFER Cause:CAUSE_NORMAL GC1: CallCtlTermConnHeldEv TERMB REASON = REASON_TRANSFER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Action	Events
	GC2: ConsultCallActive REASON = NORMAL Cause:CAUSE_NEW_CALL GC2: ConnCreatedEv B REASON = NORMAL Cause:CAUSE_NORMAL GC2: ConnConnectedEv B REASON = NORMAL Cause:CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN GC1: CallCtlConnDisconnectedEv B REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC1: TermConnDroppedEv TERMB REASON = REASON_TRANSFER Cause: CAUSE_UNKNOWN GC1: CallCtlTermConnDroppedEv TERMB REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: ConnDisconnectedEv A REASON = REASON_TRANSFER GC1: CallCtlConnDisconnectedEv A REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: CallInvalidEv REASON = REASON_TRANSFER GC2: ConnDisconnectedEv C REASON = REASON_TRANSFER GC2: CallCtlConnDisconnectedEv C REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: TermConnDroppedEv TERMB REASON = REASON_TRANSFER GC2: CallCtlTermConnDroppedEv TERMB REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER

Action	Events
	GC2: ConnDisconnectedEv B REASON = REASON_TRANSFER GC2: CallCtlConnDisconnectedEv B REASON = REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: CallInvalidEv REASON = REASON_TRANSFER GC2: CallObservationEndedEv REASON = NORMAL Cause:CAUSE_NORMAL GC1 CiscoTransferEndEv REASON = REASON_TRANSFER Cause: CAUSE_NORMAL GC2 CallObservationEndedEv REASON = NORMAL Cause:CAUSE_NORMAL

Scenario Two

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

Action	Events
Events delivered to call observer on A when call is parked. When call is unparked using GC2	

Action	Events
	GC1: ConnDisconnectedEv B REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnCreatedEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: ConnInProgressEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnQueuedEv 9999 REASON = REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: CiscoCallChangedEv Surviving = GC2 origcall = GC1 address = A REASON = REASON_UNPARK CallActiveEv REASON = REASON_UNPARK Cause: CAUSE_NEW_CALL GC2: ConnCreatedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: TermConnCreatedEv TERMA REASON = REASON_UNPARK GC2: TermConnActiveEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL

Action	Events
	CallControlCause: CAUSE_PARK GC1: ConnDisconnectedEv 9999 REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv 9995 REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: TermConnDroppedEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMA REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnDisconnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON = REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: CallInvalidEv REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC1: CallObservationEndedEv REASON = NORMAL Cause: CAUSE_NORMAL GC2: ConnCreatedEv C REASON = REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv C REASON = UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv C REASON = UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK

Scenario Three

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

Action	Events
Events delivered to call observer on A.	

Action	Events
	GC1: CallActiveEv REASON = NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnInitiatedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMA REASON = NORMAL GC1: TermConnActiveEv TERMA REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnTalkingEv TERMA REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnDialingEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv

Action	Events
	REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: ConnInProgressEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnAlertingEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnConnectedEv C REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

Scenario Four

A calls B, B redirects the call to C.

Action	Events
Events delivered to call observer on B.	

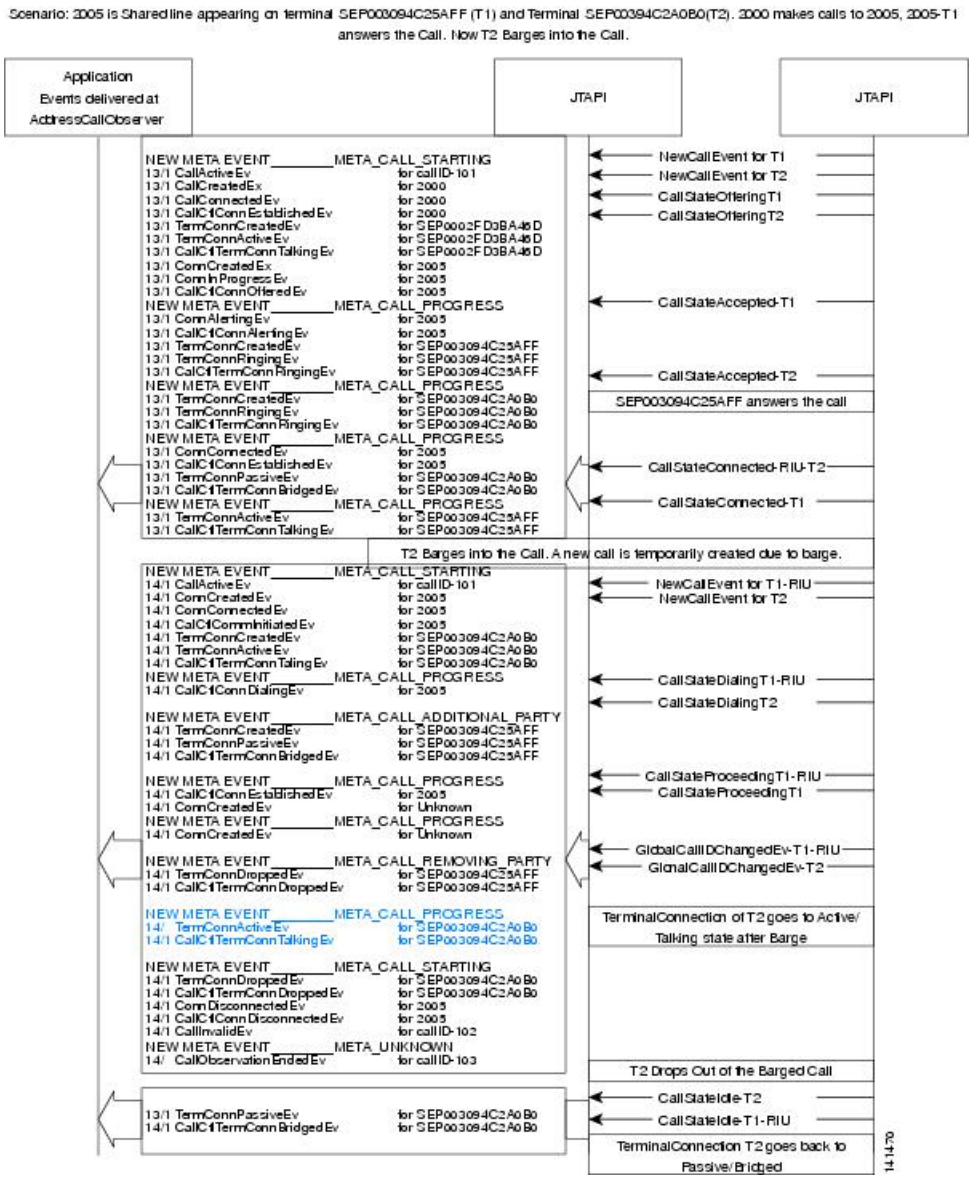
Action	Events
	GC1: CallActiveEv REASON = NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMB REASON = NORMAL Cause: Other: 0 GC1: TermConnRingingEv TERMB REASON = NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEvImpl TERMB REASON = NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnDisconnectedEv A

Action	Events
	REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: TermConnDroppedEv TERMB REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMB REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnDisconnectedEv B REASON = REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON = REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: CallInvalidEv REASON = REDIRECT Cause: CAUSE_NORMAL

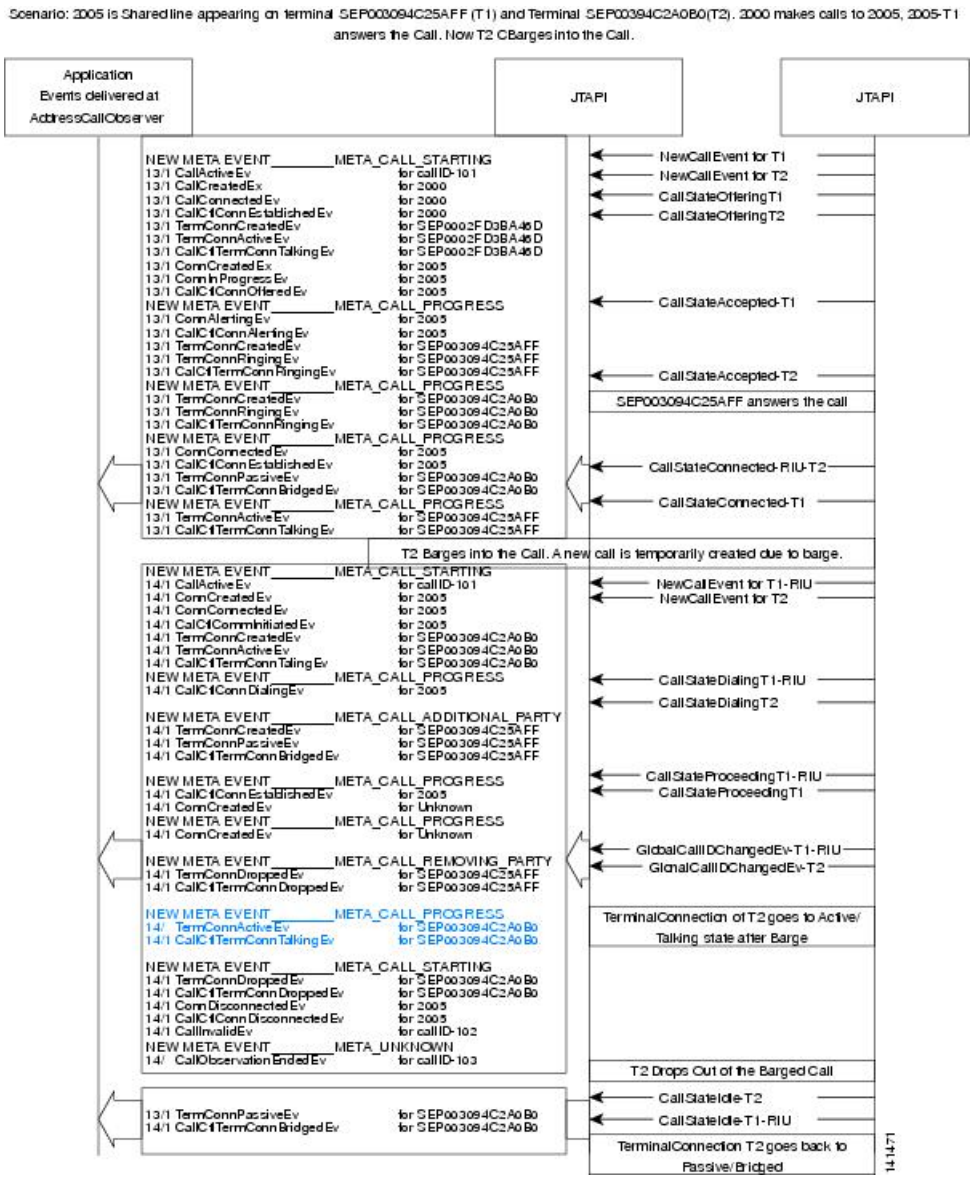
Barge and Privacy

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

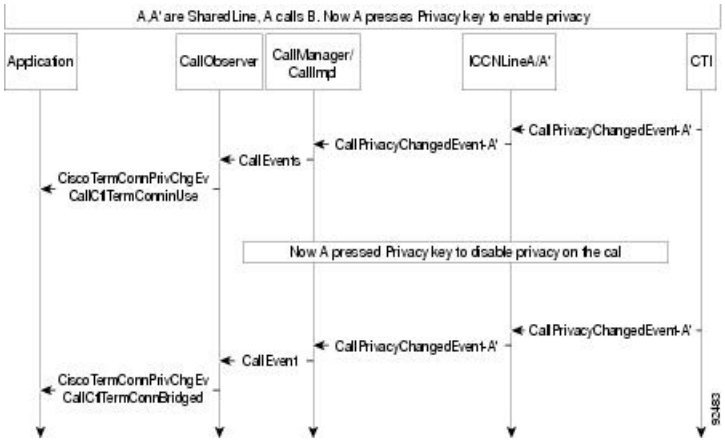
Barge


Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005. 2005-T1 answers the Call. Now T2 Barges into the Call.
Application
Events delivered at
AddressCallObserver
JTAPI
JTAPI
NEW META EVENT META_CALL_STARTING
13/1 CallActiveEv for callID-101
13/1 CallCreatedEx for 2000
13/1 CallConnectedEv for 2000
13/1 CallCtlConnEstablishedEv for 2000
13/1 TermConnCreatedEv for SEP0002FD3BA46D
13/1 TermConnActiveEv for SEP0002FD3BA46D
13/1 CallCtlTermConnTalkingEv for SEP0002FD3BA46D
13/1 ConnCreatedEx for 2005
13/1 ConnInProgressEv for 2005
13/1 CallCtlConnOfferedEv for 2005
NEW META EVENT META_CALL_PROGRESS
13/1 ConnAlertingEv for 2005
13/1 CallCtlConnAlertingEv for 2005
13/1 TermConnCreatedEv for SEP003094C25AFF
13/1 TermConnRingingEv for SEP003094C25AFF
13/1 CallCtlTermConnRingingEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnCreatedEv for SEP003094C2A0B0
13/1 TermConnRingingEv for SEP003094C2A0B0
13/1 CallCtlTermConnRingingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 ConnConnectedEv for 2005
13/1 CallCtlConnEstablishedEv for 2005
13/1 TermConnPassiveEv for SEP003094C2A0B0
13/1 CallCtlTermConnBridgedEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnActiveEv for SEP003094C25AFF
13/1 CallCtlTermConnTalkingEv for SEP003094C25AFF
NEW META EVENT META_CALL_STARTING
14/1 CallActiveEv for callID-101
14/1 ConnCreatedEv for 2005
14/1 ConnConnectedEv for 2005
14/1 CallCtlConnInitiatedEv for 2005
14/1 TermConnCreatedEv for SEP003094C2A0B0
14/1 TermConnActiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnTalingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
14/1 CallCtlConnDialingEv for 2005
NEW META EVENT META_CALL_ADDITIONAL_PARTY
14/1 TermConnCreatedEv for SEP003094C25AFF
14/1 TermConnPassiveEv for SEP003094C25AFF
14/1 CallCtlTermConnBridgedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 CallCtlConnEstablishedEv for 2005
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_PROGRESS
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_REMOVING_PARTY
14/1 TermConnDroppedEv for SEP003094C25AFF
14/1 CallCtlTermConnDroppedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 TermConnActiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnTalkingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_STARTING
14/1 TermConnDroppedEv for SEP003094C2A0B0
14/1 CallCtlTermConnDroppedEv for SEP003094C2A0B0
14/1 ConnDisconnectedEv for 2005
14/1 CallCtlConnDisconnectedEv for 2005
14/1 CallInvalidEv for callID-102
NEW META EVENT META_UNKNOWN
14/1 CallObservationEndedEv for callID-103
13/1 TermConnPassiveEv for SEP003094C2A0B0
14/1 CallCtlTermConnBridgedEv for SEP003094C2A0B0
NewCallEvent for T1
NewCallEvent for T2
CallStateOfferingT1
CallStateOfferingT2
CallStateAccepted-T1
CallStateAccepted-T2
SEP003094C25AFF answers the call
CallStateConnected-RIU-T2
CallStateConnected-T1
T2 Barges into the Call. A new call is temporarily created due to barge.
NewCallEvent for T1-RIU
NewCallEvent for T2
CallStateDialingT1-RIU
CallStateDialingT2
CallStateProceedingT1-RIU
CallStateProceedingT1
GlobalCallIDChangedEv-T1-RIU
GlobalCallIDChangedEv-T2
TerminalConnection of T2 goes to Active/
Talking state after Barge
T2 Drops Out of the Barged Call
CallStateIdle-T2
CallStateIdle-T1-RIU
TerminalConnection T2 goes back to
Passive/Bridged
141470

CBarge


Scenario: 2005 is Shared line appearing on terminal SEP003094C25AFF (T1) and Terminal SEP00394C2A0B0(T2). 2000 makes calls to 2005, 2005-T1 answers the Call. Now T2 CBarges into the Call.
Application Events delivered at AddressCallObserver
JTAPI
JTAPI
NEW META EVENT META_CALL_STARTING
13/1 CallActiveEv for callID-101
13/1 CallCreatedEx for 2000
13/1 CallConnectedEv for 2000
13/1 CallC1ConnEstablishedEv for 2000
13/1 TermConnCreatedEv for SEP0002FD3BA46D
13/1 TermConnActiveEv for SEP0002FD3BA46D
13/1 CallC1TermConnTalkingEv for SEP0002FD3BA46D
13/1 ConnCreatedEx for 2005
13/1 ConnInProgressEv for 2005
13/1 CallC1ConnOfferedEv for 2005
NEW META EVENT META_CALL_PROGRESS
13/1 ConnAlertingEv for 2005
13/1 CallC1ConnAlertingEv for 2005
13/1 TermConnCreatedEv for SEP003094C25AFF
13/1 TermConnRingingEv for SEP003094C25AFF
13/1 CalC1TermConnRingingEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnCreatedEv for SEP003094C2A0B0
13/1 TermConnRingingEv for SEP003094C2A0B0
13/1 CallC1TermConnRingingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 ConnConnectedEv for 2005
13/1 CallC1ConnEstablishedEv for 2005
13/1 TermConnPassiveEv for SEP003094C2A0B0
13/1 CallC1TermConnBridgedEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
13/1 TermConnActiveEv for SEP003094C25AFF
13/1 CallC1TermConnTalkingEv for SEP003094C25AFF
NEW META EVENT META_CALL_STARTING
14/1 CallActiveEv for callID-101
14/1 ConnCreatedEv for 2005
14/1 ConnConnectedEv for 2005
14/1 CalC1ConnInitiatedEv for 2005
14/1 TermConnCreatedEv for SEP003094C2A0B0
14/1 TermConnActiveEv for SEP003094C2A0B0
14/1 CallC1TermConnTalingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_PROGRESS
14/1 CallC1ConnDialingEv for 2005
NEW META EVENT META_CALL_ADDITIONAL_PARTY
14/1 TermConnCreatedEv for SEP003094C25AFF
14/1 TermConnPassiveEv for SEP003094C25AFF
14/1 CallC1TermConnBridgedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/1 CallC1ConnEstablishedEv for 2005
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_PROGRESS
14/1 ConnCreatedEv for Unknown
NEW META EVENT META_CALL_REMOVING_PARTY
14/1 TermConnDroppedEv for SEP003094C25AFF
14/1 CallC1TermConnDroppedEv for SEP003094C25AFF
NEW META EVENT META_CALL_PROGRESS
14/ TermConnActiveEv for SEP003094C2A0B0
14/1 CallC1TermConnTalkingEv for SEP003094C2A0B0
NEW META EVENT META_CALL_STARTING
14/1 TermConnDroppedEv for SEP003094C2A0B0
14/1 CallC1TermConnDroppedEv for SEP003094C2A0B0
14/1 ConnDisconnectedEv for 2005
14/1 CallC1ConnDisconnectedEv for 2005
14/1 CallInvalidEv for callID-102
NEW META EVENT META_UNKNOWN
14/ CallObservationEndedEv for callID-103
13/1 TermConnPassiveEv for SEP003094C2A0B0
14/1 CallC1TermConnBridgedEv for SEP003094C2A0B0
NewCallEvent for T1
NewCallEvent for T2
CallStateOfferingT1
CallStateOfferingT2
CallStateAccepted-T1
CallStateAccepted-T2
SEP003094C25AFF answers the call
CallStateConnected-RIU-T2
CallStateConnected-T1
T2 Barges into the Call. A new call is temporarily created due to barge.
NewCallEvent for T1-RIU
NewCallEvent for T2
CallStateDialingT1-RIU
CallStateDialingT2
CallStateProceedingT1-RIU
CallStateProceedingT1
GlobalCallIDChangedEv-T1-RIU
GlobalCallIDChangedEv-T2
TerminalConnection of T2 goes to Active/ Talking state after Barge
T2 Drops Out of the Barged Call
CallStateIdle-T2
CallStateIdle-T1-RIU
TerminalConnection T2 goes back to Passive/Bridged
141471

Privacy



Call Control Discovery

Scenario 1: A Calls 1000 in Other Cluster (SAF ICT)

Action	Result	Call info
A dials 1000, this call is first be intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnDialingEv - ATermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv - TA	
Called Party is 1000	ConnCreatedEv 1000 ConnInProgressEv 1000 CallCtlConnOfferedEv 1000	getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = 1000

Scenario 2: A Calls B, Within Same Cluster. B Redirects the Call to 1000, Which Is in Another Cluster (SAF ICT)

Action	Result	Call info
A calls B within the same cluster	CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEvImpl TB	
B redirects the call to 1000	TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv 1000	getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = B

Scenario 3: A Calls 1000 Which Is in the Other Cluster (SAF ICT Bandwidth Is Low)

Action	Result	Call info
A dials 1000, this call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnDialingEv - ATermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv - TA	

Action	Result	Call info
SIP trunk rejects this as bandwidth is not available	CallCtlConnEstablishedEv –A ConnCreatedEv 1000 ConnConnectedEv 1000 CallCtlConnOfferedEv 1000	getCallingAddress() = A getCalledAddress() = 1000 getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getLastRedirectedAddress() = “”
CCD Requesting feature starts PSTN failover by directing this caller to 1000’s PSTN failover number. Call is sent out to a PSTN gateway, and calling side moves to Ringback state.	CallCtlConnNetworkReachedEv 1000 CallCtlConnNetworkAlertingEv 1000	CiscoFeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCallingAddress() = A getCalledAddress() = 1000 getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getLastRedirectedAddress() = 1000

Scenario 4: A Calls B Within the Cluster. B Redirects the Call to 1000 (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B within the same cluster	CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEvImpl TB	

Action	Result	Call info
B redirects the call to 1000. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk	TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number. Call is sent out to a PSTN gateway	ConnCreatedEv 1000 ConnConnectedEv 1000	getCallingAddress() = A getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000 Reason = REASON_SAF_CCD_PSTN_FAILOVER

Scenario 5: A Calls B, B Transfers the Call to 1000 (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB	

Action	Result	Call info
B makes a consult call to 1000. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk.	GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 CallCtlConnEstablishedEv B GC2 ConnCreatedEv 1000 GC2 ConnCreatedEv 1000 GC2 CallCtlConnOfferedEv 1000	
SIP trunk rejects this call as bandwidth is not available CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number (or as configured on the server). Call is sent out to a PSTN gateway.	GC2 CallCtlConnNetworkReachedEv 1000 GC2 CallCtlConnNetworkAlertingEv 1000	CiscofeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000

Action	Result	Call info
B completes the transfer	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoTransferStartedEv GC2 CiscoCallChangedEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv 1000 GC1 ConnAlertingEv 1000 GC1 CallCtlConnAlertingEv 1000 GC1 TermConnCreatedEv 1000 GC1 TermConnRingingEv 1000 GC1 CallCtlTermConnRingingEvImpl 1000 GC2 TermConnDroppedEv 1000 GC2 CallCtlTermConnDroppedEv 1000 GC2 ConnDisconnectedEv1408972 1000 GC2 CallCtlConnDisconnectedEv 1000 GC1 TermConnDroppedEv B GC1 CallCtlTermConnDroppedEv B GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectecEv B GC2 TermConnDroppedEv B GC2 CallCtlTermConnDroppedEv B GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectecEv B GC2 CallInvalidEv GC1 CiscoTransferEndEv	Reason = REASON_TRANSFEREDCALL
A and 1000 come in direct call	GC1 ConnConnectedEv 1000 GC1 CallCtlConnEstablishedEv 1000 GC1 termConnActiveEv 1000 GC1 CallCtlTermConnTalkingEv 1000	

Scenario 6: A Calls B, B Consults 1000 and Adds It to Conference (Low Bandwidth SAF ICT)

Action	Result	Call info
A calls B	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB	
B makes a consult call to 1000 for conference. This call is first intercepted by CCD Requesting Feature, and CCD Requesting feature extends this call to SIP trunk.	GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 CallCtlConnEstablishedEv B GC2 ConnCreatedEv 1000 GC2 ConnCreatedEv 1000 GC2 CallCtlConnOfferedEv 1000	

Action	Result	Call info
SIP trunk rejects this call as no more bandwidth is available CCD Requesting feature starts PSTN failover by directing this caller to 1000's PSTN failover number; call is sent out to a PSTN gateway.	GC2 CallCtlConnNetworkReachedEv 1000 GC2 CallCtlConnNetworkAlertingEv 1000	CiscofeatureReason = NORMAL CallCtlCause = CAUSE_NORMAL getCurrentCallingAddress() = A getCurrentCalledAddress() = 1000 getCalledAddress() = B getLastRedirectedAddress() = 1000
B completes the conference	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoConferenceStartedEv GC2 termConnDroppedEv B GC2 CallCtlTermConnDroppedEv B Gc2 ConnDisconnectedEv B GC2 CallCtlConnDisConnectedEv B GC1 CallCtlTermConnTalkingEv B GC2 CiscoCallChangedEv GC1 ConnCreatedEv 1000 GC1 ConConnectedEv 1000 GC1 CallCtlConnEstablishedEv 1000 GC1 TermConnCreatedEv 1000 GC1 TermConnActiveEv 1000 GC1 CallCtlTermConnTalkingEv 1000 GC2 TermConnDroppedEv 1000 GC2 CallCtlTermConnDroppedEv 1000 GC2 ConnDisconnectedEv 1000 GC2 CallCtlConnDisconnectedEv 1000 GC2 CallInvalidEv GC1 CiscoTermConnSelectChangedEv B GC1 CiscoTermConnSelectChangedEv B	Reason = REASON_CONFERENCE

CallFwdAll Keys Press Notification

(Scenario 1): Application Is Observing A; A Goes Off-Hook

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	
A goes off-hook.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

(Scenario 2): A Goes Off-Hook; Application Starts Observing A

Action	Result	Call info
A goes off-hook	No Event is delivered	
Application starts observing A	CiscoAddrInServiceEv – A GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_SNAPSHOT

(Scenario 3): Application Is Observing A; User Presses CFwdAll Soft Key on Phone A in On-Hook State

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	

Action	Result	Call info
User presses CFwdAll soft key on phone A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

(Scenario 4): User Presses CFwdAll Soft Key on Phone A Goes in On-Hook State; Application Starts Observing A

Action	Result	Call info
User presses CFwdAll soft key on phone A	No event is delivered	
Application starts observing A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_SNAPSHOT

(Scenario 5): Application Is Observing A; A Goes Off-Hook and Presses CFwdAll Soft Key

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	

Action	Result	Call info
A goes off-hook.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall_CFWD_ALL_NONE currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
A presses CFwdAll soft key	No Event is delivered	

(Scenario 6): Application Is Observing A; User Presses CFwdAll Key on Phone A and Dial 9999(B) to Set the CFA Destination as B; User Then Presses CFwdAll Soft Key Again to Cancel the CallFwdAll

Action	Result	Call info
Application observes A.	CiscoAddrInServiceEv – A	
User presses CFwdAll soft key on phone A	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv – A TermConnCreatedEv – TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA	TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_SET currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
User dials B to set CFA destination as B	GC1: CallCtlConnDialingEv – A CallCtlConnEstablishedEv – A TermConnDroppedEv – TA CallCtlTermConnDroppedEv – TA ConnDisconnectedEv – A CallCtlConnDisconnectedEv – A CallInvalidEv	currentCalling = A currentCalled = null currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Action	Result	Call info
User presses CFwdAll soft key on phone A to cancel CFA	GC2: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - A TermConnCreatedEv - TA TermConnActiveEv –TA CallCtlTermConnTalkingEv –TA TermConnDroppedEv – TA CallCtlTermConnDroppedEv – TA ConnDisconnectedEv – A CallCtlConnDisconnectedEv – A CallInvalidEv	(GC2)TermConnActiveEv-TA. getCall().getCFWDAllKeyPressIndicator() returns CiscoCall.CFWD_ALL_CLEAR currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

Call Recording for SIP or TLS Authenticated calls

Scenario One

Recording behavior for an authenticated Phone when Service Parameter **Authenticated Phone Recording** set to **Do not Allow Recording**.

B is an Authenticated Phone having selective recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

Action	Events	Call information
termConnB.startRecording()	Recording fails with PlatformException	PlatformException.getErrorCode= CiscoRmCTIErr_SECURITY_CAPABILITY_MISMATCH

Scenario Two

Recording behavior for an authenticated Phone when Service Parameter **Authenticated Phone Recording** set to **Allow Recording**.

B is an Authenticated Phone having selective recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

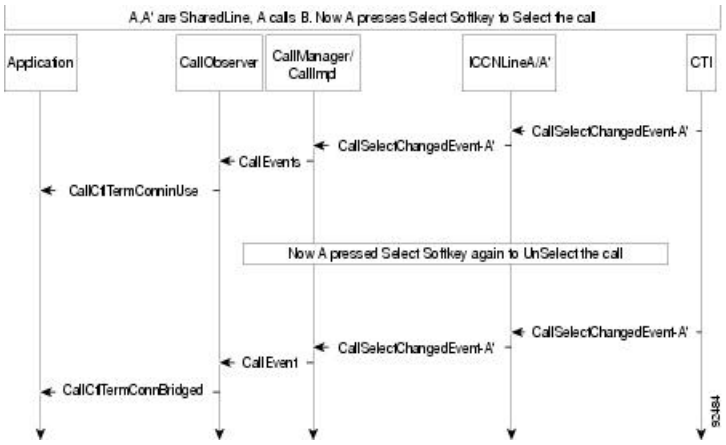
Action	Events	Call information
termConnB.startRecording()	Along with the regular events for call answer, the following events will also be delivered to the call observer: CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL CiscoTermConnRecordingTargetInfoEv CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv TA Cause: CAUSE_NORMAL	Calling: A Called: B

B is an Authenticated Phone having auto recording configured and Recording Profile assigned to it. Caller A calls B. B answers the call.

Action	Events	Call Information
When B answers	Along with the regular events for call answer, the following events will also be delivered to the call observer: CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL CiscoTermConnRecordingTargetInfoEv CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv TA Cause: CAUSE_NORMAL	Calling: A Called: B

CallSelect and UnSelect

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



Cius Persistency

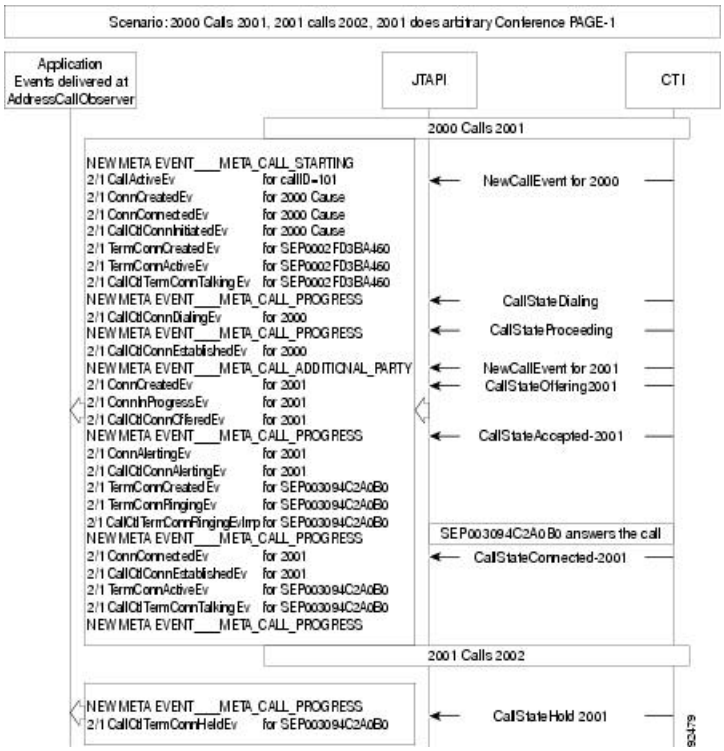
Use Cases for Cius Persistency

Usecase	Events on Provider observer	Info
Application has a wireless device TermA in its control list which is registered with IPv4 address 1.1.1.1	ProvInServiceEv	((CiscoTerminal)(Provider.getTerminal(TermA))).getIPV4Address() = 1.1.1.1
The device moves from one WiFi N/W to another resulting in the change in the IPv4 address from 1.1.1.1 to 2.2.2.2	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV4 Ev.getIPV4Address() = 2.2.2.2 ((CiscoTerminal)(Ev.getTerminal()).getIP4Address() = 2.2.2.2
The deivce moves from a IPv4 n/w to a Ipv6 n/w With new ip as 1::1	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV6 Ev.getIPV6Address() = 1::1 ((CiscoTerminal)(Ev.getTerminal()).getIP6Address() = 1::1
The Device is docked on a base station connected to the ethernet resulting in a change in IP address to 3.3.3.3	CiscoProvTerminalIPAddressChangedEv TermA	Ev.getIPAddressingMode() = CiscoTerminal.IP_ADDRESSING_MODE_IPV4 Ev.getIPV4Address() = 3.3.3.3 Ev.getTerminal() = TermA ((CiscoTerminal)(Ev.getTerminal()).getIP4Address() = 3.3.3.3

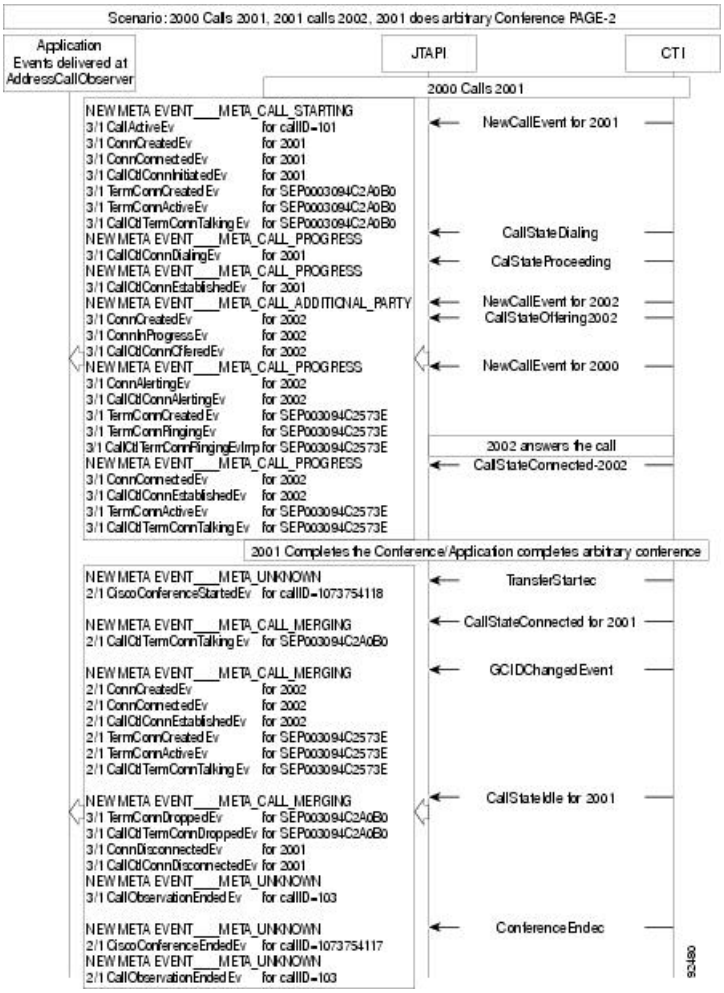
Conference and Join

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

Join/Arbitrary Conference



Join/Arbitrary Conference—Page 2



Consult Conference

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

Join Across Lines with Enhancements

The message flows for Join Across Lines with Enhancements are described in following tables. A, C, D, E and F are addresses on different terminals. B1 and B2 are addresses on the same terminal, TermB.

Action	Events
Application conferences the two calls on B1 and B2 by invoking GC1.conference(GC2) to chain two conference calls.	**Events to CallObserver of A, C and B1:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1
	Event for CallObserver at B2, D & E: ConnDisconnectedEv B2 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection of Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 & Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 & GC2

Action	Events
Application invokes GC2.conference (GC1) to chain two conference calls.	**Event for CallObserver at B2, D & E:** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2
	Events for CallObservers at A, B1 & C: ConnDisconnectedEv B1 GC1 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1

Action	Events
A, B1, C are in conference-1 (GC1), B1, D, E are in conference-2 (GC2), B2, F, G are in conference-3 (GC-3) Application completes conference at C by initiating GC1.conference(GC2, GC3) setting B1 as controller.	**Event for CallObserver at A, B1 & C:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC1 Ev.getAddedConnection will return connection for Conference-3 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 & Conference-3 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 & GC3

Action	Events
	Event for CallObserver at B1, D & E: ConnDisconnectedEv B1 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1-GC2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2
	Event for CallObserver at B2, F & G: ConnDisconnectedEv B2 GC3 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv - GC3 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC3

Action	Events
Application sets the requestor as B2 and calls GC2.conference(GC1) getControllerAddress() returns B2. getOriginalControllerAddress() returns B1.	A CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
If application uses B1 as request controller in the above setup getControllerAddress() returns B1. getOriginalControllerAddress() returns B1.	Events are same as above

CTI Manager Redundancy Handling with Least Priority CTIManager Configured

Identify a CTIManager as least priority:

Application can mark one of the CTIManagers in the initial CTIManager redundancy group or configure a new one (not part of the initial group) by invoking setLeastPriorityCtiServer().

CTI Manager Redundancy Handling with Least Priority CTI Server Set

Scenario 1: Set least priority without specifying fallback Initiation time

1. Start application and set a CTIManager as least priority. Assume CTIManager redunancy list is CT1,CTI2,CTI3.
2. Application loses connectivity to CTI1.
3. Application loses connectivity to CTI3.
4. CTI1 is reachable now.
5. Fallback is started 5 min from now if a CTI server is reachable post it.
6. Post 5 min, CTI1 is still reachable.

Action	Events
Application invokes CiscoProvider.setLeastPriorityCtiServer(CTI2).	
Application connects to CTI3.	
Application connects to CTI2.	CiscoProvConnToLeastPriorCtiServerEv
JTAPI is able to identify CTI1 reachability.	CiscoProvPrimNwReachableEv
JTAPI initiates application fallback to CTI1	Once connected to CTI1, JTAPI delivers CiscoProvFallbackToPrimNwCompltdEv event

Scenario 2: Application initiates a forced fallback

1. Start application and set a CTIManager as least priority. Assume CTIManager redunancy list is CT1,CTI2,CTI3.
2. Application loses connectivity to CTI1.
3. Application loses connectivity to CTI3.
4. CTI1 is reachable now.
5. Application monitors if CTI2 is reachable now.

Action	Events	Result
Application invokes CiscoProvider.setLeastPriorityCtiServer(CTI2,600) where 600 is the fallback initiation time.		
Application connects to CTI3.		

Action	Events	Result
Application connects to CTI2.	CiscoProvConnToLeastPriorCtiServerEv	
JTAPI is able to identify CTI1 reachability.	CiscoProvPrimNwReachableEv delivered Application queries CiscoProvPrimNwReachableEv. getReachableCtiServers() returns CTI1	
Application invokes CiscoProvider.isCtiServerAvailable(CTI2)		JTAPI return true if CTI2 was reachable now.
Application invokes CiscoProvider.initiateFallback(CTI2)		JTAPI initiates fallback to CTI2 if reachable. CiscoProvFallbackTo PrimNwCompltdEv event is returned if fallback was successful.

CTI Remote Device

Use Cases

- Group 1: Get/Add/Remove/Update on Remote Destinations
- Group 2: CTIRD Incoming/Outgoing/Disconnect/Redirect/Hold/Resume and shared-line call scenarios)
- Group 3 (CUCSF registration and unregistration, for Normal SIP mode <-> Extend mode, and terminal switching scenarios
- Group 4: Set/Reset Active Remote Destination scenarios
- Group 5: CTIRD Transfer/Conference/Multiple-Calls call scenarios
- Group 6: CTIRD URI-Dialing basic Incoming & Outgoing DVO call scenarios

CTI Remote Device Use Cases Group 1

Scenario 1-1 (Expose All RDs Information on a CTI Remote Device to Application)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermA.		TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Use Cases Group 1: Get/Add/Remove/Update on Remote Destinations

Pre-conditions on Use Cases group 1 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "CTIRD-A", Line A (DN: 1000))
- Remote Destination 1 (Name: "RD1-A", Number: "4081001111", Active RD: true)
- Remote Destination 2 (Name: "RD2-A", Number: "4081002222", Active RD: false)
- Device B (IP Phone - Name: "SEP000DED47D023", Line B (DN: 2000)
- Device C (CTI Remote Device - Name: "CTIRD-C", Line C (DN: 3000))
- No Remote Destination configured.

Scenario 1-2 (Expose Active RDs Information on a CTI Remote Device to Application)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.getActiveRemoteDestinations() on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getActiveRemoteDestinations() on TermA.		TermA.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true

Scenario 1-3 (Fetch RD Information on a CTI Remote Device That Has No RD Configured)

User1 has "CTI Remote Device C" in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal C.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermC.		TermC.getAllRemoteDestinations() = null.

Scenario 1-4 (Fetch RD Information on a 'Non-CTI Remote Device')

User1 has "Device B" IP Phone in the control list. User invokes CiscoRemoteTerminal.getAllRemoteDestinations() on terminal B.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermB.		TermB.getAllRemoteDestinations() = null.

Scenario 1-5 (Fetch Active RD Information on a CTI Remote Device That Has No Active RD Configured)

User1 has "CTI Remote Device C" in the control list. User invokes CiscoRemoteTerminal.getActiveRemoteDestinations() on terminal C.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. getAllRemoteDestinations() on TermC.		TermC.getActiveRemoteDestinations() = null.

Scenario 1-6 (Set a Non-Active RD as a New Active RD on a 'CTI Remote Device', Where There Is Already an Existing Active RD for This Device)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.setActiveRemoteDestination("4081002222", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("4081002222", true) on TermA.	CiscoProvTerminal RemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminal RemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = true

Scenario 1-7 (Add a New Non-Active RD on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("RD3-A", "4081003333", false) on TermA.	CiscoProvTerminal RemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false

Scenario 1-8 (Add a New Active RD on a 'CTI Remote Device', with Another Existing Active RD)

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("RD3-A", "4081003333", true) on TermA.	CiscoProvTermina lRemoteDestination ChangedEv	CiscoProvTerminalRemoteDestinationChangedEv. getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = true

Scenario 1-9 (Add a New RD on a 'CTI Remote Device' with a Number That Is the Same as Another Existing RD's Number)

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination("RD3-A", "4081003333", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("AnyName", "4081002222", false) on TermA.	Caught exception: com.cisco.jtapi. PlatformException Impl: Duplicated Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_DUPLICATED_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-10 (Remove a RD From a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes removeRemoteDestination("4081002222") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeRemoteDestination ("4081002222") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true

Scenario 1-11 (Remove All RD(s) From a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes removeAllRemoteDestinations() on terminal A.

Note JTAPI will loop through the terminal/device's existing remote destinations one by one, so the total number of CiscoProvTerminalRemoteDestinationChangedEv sent to an application should be the same number of available remote destinations being removed. And the order and content of each event can vary, depending on how each remote destination is stored in JTAPI's local cache RD list.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemote Destinations() on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false
	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = null.

Scenario 1-12 (Update a RD Name on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName ("4081001111", "MyHome") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4081001111", "MyHome") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-13 (Update a RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationNumber ("4081001111", "6268210080") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4081001111", "6268210080") on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "6268210080" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-14 (Add a New RD with an Invalid RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes addRemoteDestination ("iPhone5", "IAmNotANumber", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("iPhone5", "IAmNotANumber", true) on TermA.	Caught exception: com.cisco.jtapi. PlatformExceptionImpl: Invalid Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-15 (Update RD Name with an Invalid/Not-Associated RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName ("4085268222", "MyBossOffice") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name ("4085268222", "MyBossOffice") on TermA.	Caught exception: com.cisco.jtapi. PlatformExceptionImpl: Invalid Remote Destination Number	Let 'ex' be an instanceof PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_REMOTE_DESTINATION_NUMBER. TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-16 (Update RD Name with a Null RD Number on a 'CTI Remote Device')

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestinationName (null, "MyBossOffice") on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Name (null, "MyBossOffice") on TermA.	Caught exception: com.cisco.jtapi. InvalidArgument ExceptionImpl: Invalid Remote Destination Number/Name (updateRemoteDestination Name parameter).	TermA.getAllRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = true CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Scenario 1-17 (Clear an Existing Active RD as a Non-Active RD on a 'CTI Remote Device')

Explicit Pre-condition: (RD1-A: "4081001111", True; RD2-A: "4081002222", False; RD3-A: "4081003333", False)

User1 has "CTI Remote Device A" in the control list. User invokes CiscoRemoteTerminal.setActiveRemoteDestination("4081001111", false) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("4081001111", false) on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD1-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false

Scenario 1-18 (Remove All RD(s) From a 'CTI Remote Device')

User1 Has "CTI Remote Device C" in the Control List. User Invokes removeAllRemoteDestinations() on Terminal C.

Action	Events	Call info5
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemoteDestinations() on TermC.		**Note** Nothing is removed as there is no RD on this device. JTAPI won't be sending any request to CTI. No CiscoJtapiException will be thrown either.

Scenario 1-19 (Remove All 5 RD(s) From a 'CTI Remote Device')

Explicit Pre-condition: RD1-A: "4081001111", true; RD2-A: "4081002222", false; RD3-A: "4081003333", false; RD4-A: "4081004444", false; RD5-A: "4081005555", false.

User1 has "CTI Remote Device A" in the control list. User invokes removeAllRemoteDestinations() on terminal A.

Note that JTAPI will loop through the terminal/device's existing remote destinations one by one, so the total number of CiscoProvTerminalRemoteDestinationChangedEv sent to an application should be the same number

of available remote destinations being removed. And the order and content of each event can vary, depending on how each remote destination is stored in JTAPI's local cache RD list.

Also note currently there is no checking in JTAPI to limit only up to 5 RDs per CTI Remote Device. If application tries to add a new RD to an existing CTI Remote Device that already has 5 RDs, JTAPI will simply send the add request to CTI and let it decide on pass/fail.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. removeAllRemote Destinations() on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[4]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false CiscoRemoteDestinationInfo[3].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[3].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[3].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[3]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD3-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081003333" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false CiscoRemoteDestinationInfo[2].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[2].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[2].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD4-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081004444" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "RD5-A" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081005555" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = null.

Scenario 1-20 (Update a RD's Name and Number and Set It as ActiveRD on a 'CTI Remote Device' at the Same Time)

User1 has "CTI Remote Device A" in the control list. User invokes updateRemoteDestination ("4081002222", "MyVacationHome", "4081009999", true) on terminal A.

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("4081002222", "MyVacationHome", "4081009999", true) on TermA.	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "RD2-A" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false
	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "MyVacationHome" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081002222" CiscoRemoteDestinationInfo[1].getIsActiveRD() = false

Action	Events	Call info
	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemoteDestinationChangedEv.getRemoteDestinations() = CiscoRemoteDestinationInfo[2]. CiscoRemoteDestinationInfo[0].getRemoteDestinationName() = "MyHome" CiscoRemoteDestinationInfo[0].getRemoteDestinationNumber() = "4081001111" CiscoRemoteDestinationInfo[0].getIsActiveRD() = false CiscoRemoteDestinationInfo[1].getRemoteDestinationName() = "MyVacationHome" CiscoRemoteDestinationInfo[1].getRemoteDestinationNumber() = "4081009999" CiscoRemoteDestinationInfo[1].getIsActiveRD() = true

CTI Remote Device Use Cases Group 2

Use Cases Group 2: CTIRD Incoming/Outgoing/Disconnect/Redirect/Hold/Resume and Shared-Line Call Scenarios

Pre-conditions on Use Cases group 2 below with default jtapi.ini settings, unless specified explicitly. Note that the CTI Ports have Auto-Accept enabled:

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD1", Line A (DN: 8881000))
- Remote Destination 1 (Name: "IRVOffice", Number: "919498231202", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort1", Line B (DN: 8881000))
- Device C (CTI Port - Name: "irvCTIPort6", Line C (DN: 8886000))
- Device D (CTI Port - Name: "irvCTIPort7", Line C (DN: 8887000))
- Device E (CTI Remote Device - Name: "irvCTIRD2", Line E (DN: 8889000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device F (CTI Remote Device - Name: "irvCTIRD3", Line E (DN: 8889001))
- Remote Destination 1 (Name: "IRVCell2", Number: "916267829526", Active RD: true)

Scenario 2-1 (Incoming Call From CTI Port to CTI Remote Device)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-2 (Incoming Call From CTI Port to Non-Observed CTI Remote Device)

C calls E, Application is observing C only on address and terminal. No observer on E. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-3 (Incoming Call From CTI Port to CTI Remote Device, but No Answer on Remote Destination)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 does not answers the call and time out.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEv 8889000 GC1: CallObservationEndedEv	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-4 (Incoming Call From CTI Port to CTI Remote Device, and Redirect to Another CTI Port)

C calls E, and E redirects the call to D, Application is observing all C, D, E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes connection on irvCTIRD2.redirect (8887000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8887000, null, REDIRECT_WITHOUT_ MODIFIED_CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
irvCTIPort7 answers the call.	GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000

Scenario 2-5 (Incoming Call From CTI Port to CTI Remote Device, and Redirect to Another CTI Remote Device)

C calls E, and E redirects the call to F, and C redirect the call to E. Application is observing all C, E, F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes connection on irvCTIRD2.redirect(8889001, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8889001, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8886000 :: LastRedirectedPartyAddress: 8889000
User1 invokes connection on irvCTIPort6.redirect(8889000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8889000, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1)	GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889001 :: LastRedirectedPartyAddress: 8886000

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889001 :: LastRedirectedPartyAddress: 8886000

Scenario 2-6 (Incoming Call From CTI Port to CTI Remote Device with a Shared-Line of Another CTI Port)

C calls A (with a shared line with B), Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CallingAddress = 8886000, CalledAddress = 8881000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8881000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
Disconnect the call from irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-7 (Incoming Call From CTI Port to CTI Port with a Shared-Line of a CTI Remote Device)

C calls B (with a shared line of A), Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CallingAddress = 8886000, CalledAddress = 8881000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8881000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort1 answers the call.	GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnTalkingEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
Disconnect the call from irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-8 (Outgoing Call From CTI Remote Device to CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings and answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-9 (Outgoing Call From CTI Remote Device (with a Shared Line of CTI Port) to Another CTI Port)

A (with a shared line of B) calls D, Application is observing both A, B, and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD1, 8881000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000	CallingAddress = Unknown, CalledAddress = 8881000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8881000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1	
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1	CallingAddress = Unknown, CalledAddress = 8881000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8881000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8881000, No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CallingAddress = Unknown, CalledAddress = 8887000, CurrentCallingAddress = 8881000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8881000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
Disconnect the call from 8881000 connection.	GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress

Scenario 2-10 (Outgoing Call From CTI Remote Device to CTI Port, but No Answer on Active Remote Destination)

E calls D, Application is observing both E and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 does not answer the call and time out.	GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-11 (Outgoing Call From CTI Remote Device to CTI Port, but No Answer on CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8887000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8887000, No LastRedirectedPartyAddress
irvCTIPort7 does not answer the call, time out.	GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: ConnFailedEvent 8889000 GC1: CallCtlConnFailedEv 8889000 GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-12 (Outgoing Call From Non-Observed CTI Remote Device to CTI Port)

E calls D, Application is observing only on D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
From another provider, User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: Unknown :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnCreatedEvent 8889000 GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIPort7 answers the call.	GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-13 (Outgoing Call From CTI Remote Device to Non-Observed CTI Port)

E calls D, Application is observing only on E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
From another provider, User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings and answers the call from another provider.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-14 (Outgoing Call From CTI Remote Device to Another CTI Remote Device)

E calls F, Application is observing both E and F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8889001).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889000, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889000, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Scenario 2-15 (Outgoing Call From CTI Remote Device to Another CTI Remote Device, Then Redirect Again to a Third CTI Remote Device with a Shared-Line)

E calls F, then F redirect to A (with a shared-line with B), Application is observing all A, B, E and F on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8889001).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: Unknown:: No LastRedirectedPartyAddress
irvCTIRD3's Active remote destination of 916267829526 answers the call.	GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3 GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CurrentCalledAddress: 8889001 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Action	Events	Call info
User invokes connection on irvCTIRD3.redirect(8881000, REDIRECT_NORMAL, DEFAULT_SEARCH_SPACE, CALLED_ADDRESS_UNCHANGED, REDIRECT, 8881000, null, REDIRECT_WITHOUT_MODIFIED_ CALLING_PARTY, 1). Both irvCTIRD1 and irvCTIPort1 are ringing.	GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001
irvCTIRD1's Active remote destination of 919498231202 answers the call. Terminal connection of irvCTIRD1 goes to 'talking' and irvCTIPort1 goes to 'bridged'.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Action	Events	Call info
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8889001

Scenario 2-16 (Disconnect an Incoming Call on CTI Remote Device After Answer While Talking)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User invokes connection.disconnect on irvCTIRD2 while talking.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-17 (Disconnect an Incoming Call on CTI Remote Device After Answer While Talking)

C calls E, Application is observing both C and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 8889000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invokes connection.disconnect on irvCTIRD2 while it's still ringing on Active remote destination of 16267829523.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: ConnFailedEvent 8886000 GC1: CallCtlConnFailedEv 8886000 GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-18 (Disconnect an Outgoing Call From CTI Remote Device to CTI Port; Disconnect After Answering on Remote Destination and Answering on Called CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIPort7 rings and answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Action	Events	Call info
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-19 (disconnect an Outgoing Call From CTI Remote Device to CTI Port; Disconnect After Answering on Remote Destination but Before Answering on Called CTI Port)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 rings	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000
Disconnect the call from 8889000 connection.	GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv irvCTIRD2 GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8889000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-20 (Disconnect an Outgoing Call From CTI Remote Device to CTI Port; Drop the Call Before Even Answering on Remote Destination. Note That Only One Connection on CTI Remote Device Which Is in Offering State)

E calls D, Application is observing both D and E on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD2, 8889000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD2's Active remote destination of 916267829523 rings, User1 drops the call to disconnect from 8889000 connection.	GC1: ConnDisconnectedEvent 8889000 GC1: CallCtlConnDisconnectedEv 8889000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8889000 :: CurrentCallingAddress: 8889000 :: No LastRedirectedPartyAddress

Scenario 2-21 (Incoming Call From CTI Port to CTI Remote Device with a Shared-line of Another CTI Port). Note That irvCTIRD1 Remote Destination Answers the Call; Hold irvCTIRD1, Unhold irvCTIRD1; Then Hold irvCTIRD1, Unhold irvCTIPort1 Which Results irvCTIRD1 Got Disconnected; Then Hold irvCTIPort6, Unhold irvCTIPort6, Then Disconnect 8886000)

C calls A (with a shared line with B) with several hold/resume operations on different terminals. Application is observing A, B, and C on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIPort6, 8886000, 881000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnRingingEvent irvCTIRD1 GC1: CallCtlTermConnRingingEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnHeldEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Action	Events	Call info
User invoke unhold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnTalkingEv irvCTIRD1 GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1 GC1: CallCtlTermConnHeldEv irvCTIPort1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke unhold on terminalconnection of irvCTIPort1 (while it's in Bridged state). This results in irvCTIRD1 being dropped.	GC1: CallCtlTermConnTalkingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke hold on terminalconnection of irvCTIPort6	GC1: CallCtlTermConnHeldEv irvCTIPort6	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress
User invoke unhold on terminalconnection of irvCTIPort6	GC1: CallCtlTermConnTalkingEv irvCTIPort6	
Disconnect the call from connection of irvCTIPort6.	GC1: TermConnDroppedEv irvCTIPort6 GC1: CallCtlTermConnDroppedEv irvCTIPort6 GC1: ConnDisconnectedEvent 8886000 GC1: CallCtlConnDisconnectedEv8886000 GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8881000 :: CurrentCallingAddress: 8886000 :: No LastRedirectedPartyAddress

Scenario 2-22 (Outgoing Call From CTI Remote Device (with a Shared Line of CTI Port) to Another CTI Port). Note That irvCTIPort1 Rings and Remote Destination on irvCTIRD1 Rings, Remote Destination Answers the Call, Call Is Then Offered on irvCTIPort7. After Answer on irvCTIPort7, Disconnect From 8881000 Connection)

A (with a shared line of B) calls D, then with several hold/unhold operations at different terminals. Application is observing both A, B, and D on addresses and terminals. GC1 is the GCID of the call.

Action	Events	Call info
User1 invokes call.connect(irvCTIRD1, 8881000, 8887000).	GC1: CallActiveEvent GC1: ConnCreatedEvent 8881000 GC1: ConnInprogressEvent 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnAlertingEvent 8881000 GC1: CallCtlConnAlertingEv 8881000	CurrentCalledAddress: 8881000:: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnCreatedEvent irvCTIPort1 GC1: TermConnRingingEvent irvCTIPort1 GC1: CallCtlTermConnRingingEv irvCTIPort1	
irvCTIRD1's Active remote destination of 919498231202 answers the call.	GC1: ConnConnectedEvent 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnCreatedEvent irvCTIRD1 GC1: TermConnActiveEvent irvCTIRD1 GC1: CallCtlTermConnTalkingEv irvCTIRD1	CurrentCalledAddress: 8881000:: CurrentCallingAddress: 8881000:: No LastRedirectedPartyAddress
irvCTIPort7 rings	GC1: TermConnPassiveEvent irvCTIPort1 GC1: CallCtlTermConnBridgedEv irvCTIPort1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIPort7 answers the call.	GC1: ConnCreatedEvent 8887000 GC1: ConnInprogressEvent 8887000 GC1: CallCtlConnOfferedEv 8887000 GC1: ConnAlertingEvent 8887000 GC1: CallCtlConnAlertingEv 8887000 GC1: TermConnCreatedEvent irvCTIPort7 GC1: TermConnRingingEvent irvCTIPort7 GC1: CallCtlTermConnRingingEv irvCTIPort7 GC1: ConnConnectedEvent 8887000 GC1: CallCtlConnEstablishedEv 8887000 GC1: TermConnActiveEvent irvCTIPort7 GC1: CallCtlTermConnTalkingEv irvCTIPort7	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000:: LastRedirectedPartyAddress: 8887000
User invoke hold on terminalconnection of irvCTIRD1	GC1: CallCtlTermConnHeldEv irvCTIRD1 GC1: TermConnActiveEvent irvCTIPort1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000
User invoke unhold on terminalconnection of irvCTIPort1 (while it's in Bridged state). This results in irvCTIRD1 being dropped.	GC1: CallCtlTermConnHeldEv irvCTIPort1 GC1: CallCtlTermConnTalkingEv irvCTIPort1 GC1: TermConnDroppedEv irvCTIRD1 GC1: CallCtlTermConnDroppedEv irvCTIRD1	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000

Action	Events	Call info
Disconnect the call from 8881000 connection.	GC1: TermConnDroppedEv irvCTIPort1 GC1: CallCtlTermConnDroppedEv irvCTIPort1 GC1: ConnDisconnectedEvent 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: TermConnDroppedEv irvCTIPort7 GC1: CallCtlTermConnDroppedEv irvCTIPort7 GC1: ConnDisconnectedEvent 8887000 GC1: CallCtlConnDisconnectedEv 8887000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CurrentCalledAddress: 8887000 :: CurrentCallingAddress: 8881000 :: LastRedirectedPartyAddress: 8887000

Scenario 2-23 (Superprovider Acquires a CTIRD That Is Not on User Control List)

User1 open a provider which can observe any terminal (User1 with "Standard CTI Allow Control of All Devices" role), and then acquire a CTI Remote Device "CTIRD_UP" that is not on User1's control list).

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes provider.createTerminal(CTIRD_UP).	CiscoAddrCreatedEv: 8889999 CiscoTermCreatedEv: CTIRD_UP	

CTI Remote Device Use Cases Group 3

Use Cases Group 3 (CUCSF Registration and Unregistration, for Normal SIP Mode <-> Extend Mode, and Terminal Switching Scenarios)

Pre-conditions on Use Cases group 3 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state.
- Device A (CUCSF - Name: "irvCSF1", Line A (DN: 7771000))
- Remote Destination 1 (Name: "IRVCell", Number: "916267829523", Active RD: true)
- Scenario 3-1 (Registration of CUCSF in between Extend mode and SIP mode).:

Device A is registered in normal SIP mode when Webex with Jabber client is running and configured this device as its associated phone. User1 open provider and add observers on this device to bring it in service. Now exit Webex to unregister it from SIP, and then User1 calls CiscoTerminal.register() to register it to Extend mode from JTAPI, then add observers back to bring it in service. Now User1 unregister it from Extend mode in JTAPI by calling CiscoRemoteTerminal.unregister(). Now open Webex again to register this device back to SIP mode, and add observers back to bring it in service.

Action	Events	Info
Webex is opened. User1 adds provider observer. registerFeature 1235 on provider. addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: ProvInServiceEv 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = true CiscoRemoteTerminal.getType() = 503 CiscoRemoteTerminal.getTypeName() = Cisco Unified Client Services Framework
Exit Webex	Provider: CiscoProvTerminalUnRegisteredEv irvCSF1 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermOutOfServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = false
User1 calls CiscoTerminal.register() on irvCSF1. addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: CiscoAddrRemovedEv 7771000 Provider: CiscoTermRemovedEv irvCSF1 Provider: CiscoAddrCreatedEv 7771000 Provider: CiscoTermCreatedEv irvCSF1 Provider: CiscoProvTerminalRegisteredEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrOutOfServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 3 CiscoTerminal.isRegistered() = true CiscoRemoteTerminal.getRegistrationType() = 8 CiscoRemoteTerminal.isRegisteredByThisApp() = true
User1 calls CiscoRemoteTerminal. unregister()	7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermOutOfServiceEv Provider: CiscoProvTerminalUnRegisteredEv	CiscoTerminal.getProtocol = 3 CiscoTerminal. isRegistered() = false CiscoRemoteTerminal.getRegistrationType() = -1 CiscoRemoteTerminal. isRegisteredByThisApp() = false

Action	Events	Info
Open Webex addObserver on address 7771000. addObserver on terminal irvCSF1.	Provider: CiscoAddrRemovedEv 7771000 Provider: CiscoTermRemovedEv irvCSF1 Provider: CiscoAddrCreatedEv 7771000 Provider: CiscoTermCreatedEv irvCSF1 Provider: CiscoProvTerminalRegisteredEv 7771000: CiscoAddrOutOfServiceEv irvCSF1: CiscoTermInServiceEv 7771000: CiscoAddrInServiceEv irvCSF1: CiscoTermInServiceEv	CiscoTerminal.getProtocol = 2 CiscoTerminal.isRegistered() = true

CTI Remote Device Use Cases Group 4

Use Cases Group 4 (Set/Reset Active Remote Destination Scenarios)

Pre-conditions on Use Cases group 4 below with default jtapi.ini settings, unless specified explicitly:

- Provider is IN_SERVICE state. Single node.
- Device A (CTI Remote Device - Name: "irvCTIRD2", Line A (DN: 8881000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: false)
- Scenario 4-1 (User1 opens provider P1, set RD1 as active. Now User1 opens another provider P2, and set same RD1 as active again. Now stop CTI Manager service in this single node, the active RD would be clear out. Now restart CTI Manager, JTAPI will do a provider retry, and upon successfully connection, it will automatically reset the same RD1 as active again seamlessly.):

Action	Events	Info
User1 open Provider P1 and adds provider observer.	P1: ProvInServiceEv	
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P1.	P1: CiscoProvTerminalRemoteDestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true
User1 open Provider P2 and adds provider observer.	P2: ProvInServiceEv	

Action	Events	Info
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P2.	P1: CiscoProvTerminalRemote DestinationChangedEv P2: CiscoProvTerminalRemote DestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : false CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = false P2 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true
Stop CTI Manager on this single node where P1 & P2 are connected to. And this active RD will be clear out automatically from CTI/CCM side.	P1: ProvOutOfServiceEv P2: ProvOutOfServiceEv	
Start this CTI Manager. And JTAPI will automatically reset the same RD1 as active again seamlessly.	P1: ProvInServiceEv P2: ProvInServiceEv	Note that no CiscoProvTerminalRemote DestinationChangedEv will be sent to application because it is the same active RD that application previously set.

Scenario 4-2 (User1 Opens Provider P1, Add All Observers on Provider, Terminals, Addresses, Then Set RD1 as Active. Now Stop CTI Manager Service, the Active RD Would Be Clear Out. Now Restart CTI Manager, JTAPI Will Do a Provider Retry, and Upon Successfully Connection, It Will Automatically Reset the Same RD1 as Active Again Seamlessly)

Action	Events	Info
User1 open Provider P1 and adds provider observer.	P1: ProvInServiceEv	
User1 calls CiscoRemoteTerminal. setActiveRemote Destination ("916267829523", true) from P1.	P1: CiscoProvTerminalRemote DestinationChangedEv	P1 changed event: Remote Terminal: irvCTIRD2 :: [Remote Destination 1: Name:IRVCell1, Number:916267829523, IsActiveRD:true] :: IsMyAppLastToSetActiveRD : true CiscoRemoteTerminal. isMyAppLastToSetActiveRD() = true

Action	Events	Info
Stop CTI Manager where P1 is connected to. And this active RD will be clear out automatically from CTI/CCM side.	8881000:: Event: CiscoAddrOutOfServiceEv irvCTIRD2:: Event: CiscoTermOutOfServiceEv P1: ProvOutOfServiceEv	
Start this CTI Manager. And JTAPI will automatically reset the same RD1 as active again seamlessly.	P1: ProvInServiceEv irvCTIRD2:: Event: CiscoTermInServiceEv 8881000:: Event: CiscoAddrInServiceEv	Note that no CiscoProvTerminalRemote DestinationChangedEv will be sent to application because it is the same active RD that application previously set.

CTI Remote Device Use Cases Group 5

Use Cases Group 5 (CTIRD Transfer/Conference/Multiple-Calls Call Scenarios)

Pre-conditions on Use Cases group 5 below with default jtapi.ini settings, unless specified explicitly (Note: The CTI Ports have Auto-Accept enabled):

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD2", Line A (DN: 8889000))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort4", Line B (DN: 8884000))
- Device C (CTI Port - Name: "irvCTIPort5", Line B (DN: 8884000))
- Device D (CTI Port - Name: "irvCTIPort6", Line D (DN: 8886000))
- Device E (CTI Remote Device - Name: "irvCTIRD3", Line E (DN: 8889001))
- Remote Destination 1 (Name: "IRVHome1", Number: "916268210080", Active RD: true)

Scenario 5-1 (Direct Transfer on CTI Remote Device to CTI Port)

D calls A with GC1 as GCID of call; A calls B with GC2 as GCID of call. Set A as transfer controller, and then transfer call from GC2 to GC1. Application is observing all A, B, D.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIRD2, 8889000, 8884000) , and answer() on irvCTIPort4 terminal connection.	GC1: CallCtlTermConnHeldEv irvCTIRD2 GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnInitiatedEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2 GC2: CallCtlConnDialingEv : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: ConnCreatedEvent : Address: 8884000 GC2: ConnInprogressEvent : Address: 8884000 GC2: CallCtlConnOfferedEv : Address: 8884000 GC2: ConnAlertingEvent : Address: 8884000 GC2: CallCtlConnAlertingEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort4 GC2: TermConnRingingEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIPort4 GC2: TermConnCreatedEvent : Terminal: irvCTIPort5 GC2: TermConnRingingEvent : Terminal: irvCTIPort5	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8884000, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8884000, No LastRedirectedPartyAddress

Action	Events	Call info
	GC2: CallCtlTermConnRingingEv : Terminal: irvCTIPort5 GC2: ConnConnectedEvent : Address: 8884000 GC2: CallCtlConnEstablishedEv : Address: 8884000 GC2: TermConnActiveEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort4 GC2: TermConnPassiveEvent : Terminal: irvCTIPort5 GC2: CallCtlTermConnInUseEv : Terminal: irvCTIPort5	
User1 invokes GC2.setTransferController (terminal connection of irvCTIRD2).	GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2 GC1: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000 CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes GC2.transfer(GC1).	GC2: CiscoTransferStartEv GC1: TermConnDroppedEv irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC1: CiscoCallChangedEv GC2: ConnCreatedEvent : Address: 8886000 GC2: ConnConnectedEvent : Address: 8886000 GC2: CallCtlConnEstablishedEv : Address: 8886000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort6 GC2: TermConnActiveEvent : Terminal: irvCTIPort6 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort6 GC1: TermConnDroppedEv : Terminal: irvCTIPort6	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000
	GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Action	Events	Call info
disconnect() the call on 8886000 connection.	GC2: TermConnDroppedEv irvCTIPort6 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC2: ConnDisconnectedEvent : Address: 8886000 GC2: CallCtlConnDisconnectedEv : Address: 8886000 GC2: TermConnDroppedEv : Terminal: irvCTIPort4 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort4 GC2: TermConnDroppedEv : Terminal: irvCTIPort5 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort5 GC2: ConnDisconnectedEvent : Address: 8884000 GC2: CallCtlConnDisconnectedEv : Address: 8884000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8889000, CalledAddress = 8884000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8886000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8886000, LastRedirectedPartyAddress = 8889000

Scenario 5-2 (Conference Call on CTI Remote Device and CTI Port with Another CTI Remote Device)

D calls A with GC1 as GCID of call; A calls E with GC2 as GCID of call. Set A as conference controller, and then conference/join call from GC2 to GC1. Application is observing all A, D, E.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIRD2, 8889000, 8889001).	GC1: CallCtlTermConnHeldEv irvCTIRD2 GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnInitiatedEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2 GC2: CallCtlConnDialingEv : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: ConnCreatedEvent : Address: 8889001 GC2: ConnInprogressEvent : Address: 8889001 GC2: CallCtlConnOfferedEv : Address: 8889001 GC2: ConnAlertingEvent : Address: 8889001 GC2: CallCtlConnAlertingEv : Address: 8889001 GC2: TermConnCreatedEvent : Terminal: irvCTIRD3 GC2: TermConnRingingEvent : Terminal: irvCTIRD3 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIRD3	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
irvCTIRD3's Active remote destination of 916268210080 answers the call.	GC2: ConnConnectedEvent : Address: 8889001 GC2: CallCtlConnEstablishedEv : Address: 8889001 GC2: TermConnActiveEvent : Terminal: irvCTIRD3 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD3	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000
User1 invokes GC2.setConference Controller (terminal connection of irvCTIRD2).	GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2 GC2: CiscoTermConnSelectChangedEv : Terminal: irvCTIRD2	

Action	Events	Call info
User1 invokes GC2.conference(GC1).	GC2: CiscoConferenceStartEv GC1: TermConnDroppedEv : Terminal: irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC1: CiscoCallChangedEv GC2: ConnCreatedEvent : Address: 8886000 GC2: ConnConnectedEvent : Address: 8886000 GC2: CallCtlConnEstablishedEv : Address: 8886000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort6 GC2: TermConnActiveEvent : Terminal: irvCTIPort6 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort6 GC1: TermConnDroppedEv : Terminal: irvCTIPort6 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000

Action	Events	Call info
disconnect() the call on 8889000 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIRD3 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD3 GC2: ConnDisconnectedEvent : Address: 8889001 GC2: CallCtlConnDisconnectedEv : Address: 8889001	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = Unknown, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = Unknown, LastRedirectedPartyAddress: 8889000
disconnect() the call on 8889001 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIPort6 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC2: ConnDisconnectedEvent : Address: 8886000 GC2: CallCtlConnDisconnectedEv : Address: 8886000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889001, LastRedirectedPartyAddress: 8889000

Scenario 5-3 (Multiple Calls on CTI Remote Device)

D calls A with GC1 as GCID of call; B calls A with GC2 as GCID of call. Application is observing all A, B, D.

Action	Events	Call info
User1 invokes call. connect (irvCTIPort6, 8886000, 8889000)	GC1: CallActiveEvent GC1: ConnCreatedEvent 8886000 GC1: ConnConnectedEvent 8886000 GC1: CallCtlConnInitiatedEv 8886000 GC1: TermConnCreatedEvent irvCTIPort6 GC1: TermConnActiveEvent irvCTIPort6 GC1: CallCtlTermConnTalkingEv irvCTIPort6 GC1: CallCtlConnDialingEv 8886000 GC1: CallCtlConnEstablishedEv 8886000 GC1: ConnCreatedEvent 8889000 GC1: ConnInprogressEvent 8889000 GC1: CallCtlConnOfferedEv 8889000 GC1: ConnAlertingEvent 8889000 GC1: CallCtlConnAlertingEv 8889000 GC1: TermConnCreatedEvent irvCTIRD2 GC1: TermConnRingingEvent irvCTIRD2 GC1: CallCtlTermConnRingingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
irvCTIRD2's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889000 GC1: CallCtlConnEstablishedEv 8889000 GC1: TermConnActiveEvent irvCTIRD2 GC1: CallCtlTermConnTalkingEv irvCTIRD2	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
User1 invokes call. connect (irvCTIPort4, 8884000, 8889000)	GC2: CallActiveEvent GC2: ConnCreatedEvent : Address: 8884000 GC2: ConnConnectedEvent : Address: 8884000 GC2: CallCtlConnInitiatedEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort4 GC2: TermConnActiveEvent : Terminal: irvCTIPort4 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIPort4 GC2: CallCtlConnDialingEv : Address: 8884000 GC2: TermConnCreatedEvent : Terminal: irvCTIPort5 GC2: TermConnPassiveEvent : Terminal: irvCTIPort5 GC2: CallCtlTermConnInUseEv : Terminal: irvCTIPort5 GC2: CallCtlConnEstablishedEv : Address: 8884000 GC2: ConnCreatedEvent : Address: 8889000 GC2: ConnInprogressEvent : Address: 8889000 GC2: CallCtlConnOfferedEv : Address: 8889000 GC2: ConnAlertingEvent : Address: 8889000 GC2: CallCtlConnAlertingEv : Address: 8889000 GC2: TermConnCreatedEvent : Terminal: irvCTIRD2 GC2: TermConnRingingEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnRingingEv : Terminal: irvCTIRD2	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress CallingAddress = 8889000, CalledAddress = 8889001, CurrentCallingAddress = 8889000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8889000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
User1 invokes GC2.answer(terminal connection of irvCTIRD2).	GC1: CallCtlTermConnHeldEv : Terminal: irvCTIRD2 GC2: ConnConnectedEvent : Address: 8889000 GC2: CallCtlConnEstablishedEv : Address: 8889000 GC2: TermConnActiveEvent : Terminal: irvCTIRD2 GC2: CallCtlTermConnTalkingEv : Terminal: irvCTIRD2	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

Action	Events	Call info
disconnect() the GC2 call on 8889000 connection.	GC2: TermConnDroppedEv : Terminal: irvCTIRD2 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC2: ConnDisconnectedEvent : Address: 8889000 GC2: CallCtlConnDisconnectedEv : Address: 8889000 GC2: TermConnDroppedEv : Terminal: irvCTIPort4 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort4 GC2: TermConnDroppedEv : Terminal: irvCTIPort5 GC2: CallCtlTermConnDroppedEv : Terminal: irvCTIPort5 GC2: ConnDisconnectedEvent : Address: 8884000 GC2: CallCtlConnDisconnectedEv : Address: 8884000 GC2: CallInvalidEvent GC2: CallObservationEndedEv	CallingAddress = 8884000, CalledAddress = 8889000, CurrentCallingAddress = 8884000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8884000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress
User1 invokes unhold() on GC1 terminal connection of irvCTIRD2. disconnect() the GC1 call on 8889000 connection.	GC1: CallCtlTermConnTalkingEv GC1: TermConnDroppedEv : Terminal: irvCTIRD2 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIRD2 GC1: ConnDisconnectedEvent : Address: 8889000 GC1: CallCtlConnDisconnectedEv : Address: 8889000 GC1: TermConnDroppedEv : Terminal: irvCTIPort6 GC1: CallCtlTermConnDroppedEv : Terminal: irvCTIPort6 GC1: ConnDisconnectedEvent : Address: 8886000 GC1: CallCtlConnDisconnectedEv : Address: 8886000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = 8886000, CalledAddress = 8889000, CurrentCallingAddress = 8886000, CurrentCalledAddress = 8889000, ModifiedCallingAddress = 8886000, ModifiedCalledAddress = 8889000, No LastRedirectedPartyAddress

CTI Remote Device Use Cases Group 6

Use Cases Group 6 (CTIRD URI-Dialing Basic Incoming and Outgoing DVO Call Scenarios)

Pre-conditions on Use Cases group 6 below with default jtapi.ini settings, unless specified explicitly (Note: The CTI Ports have Auto-Accept enabled):

- Provider is IN_SERVICE state.
- Device A (CTI Remote Device - Name: "irvCTIRD3", Line A (DN: 8889001, Directory URIs: "8889001A@cisco.com"))
- Remote Destination 1 (Name: "IRVCell1", Number: "916267829523", Active RD: true)
- Device B (CTI Port - Name: "irvCTIPort2", Line B (DN: 8882000, Directory URIs: "8882000A@cisco.com"))

Scenario 6-1 (Basic Incoming Call From CTI Port to CTI Remote Device Via URI)

B calls A with GC1 as GCID of call. Application is observing both A and B.

Action	Events	Call info
User1 invokes call.connect (irvCTIPort2, 8882000, "8889001A@cisco.com")	GC1: CallActiveEvent GC1: ConnCreatedEvent 8882000 GC1: ConnConnectedEvent 8882000 GC1: CallCtlConnInitiatedEv 8882000 GC1: TermConnCreatedEvent irvCTIPort2 GC1: TermConnActiveEvent irvCTIPort2 GC1: CallCtlTermConnTalkingEv irvCTIPort2 GC1: CallCtlConnDialingEv 8882000 GC1: CallCtlConnEstablishedEv 8882000 GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001 GC1: ConnAlertingEvent 8889001 GC1: CallCtlConnAlertingEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnRingingEvent irvCTIRD3 GC1: CallCtlTermConnRingingEv irvCTIRD3	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress
Disconnect() the call on 8882000 connection.	GC1: TermConnDroppedEv irvCTIPort2 GC1: CallCtlTermConnDroppedEv irvCTIPort2 GC1: ConnDisconnectedEvent 8882000 GC1: CallCtlConnDisconnectedEv 8882000 GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = 8882000, CalledAddress = 8889001, CurrentCallingAddress = 8882000, CurrentCalledAddress = 8889001, ModifiedCallingAddress = 8882000, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Scenario 6-2 (Basic Outgoing DVO Call From CTI Remote Device to CTI Port Via URI)

A calls B with GC1 as GCID of call. Application is observing both A and B.

Action	Events	Call info
User1 invokes call. connect (irvCTIRD3, 8889001, "8882000A@cisco.com")	GC1: CallActiveEvent GC1: ConnCreatedEvent 8889001 GC1: ConnInprogressEvent 8889001 GC1: CallCtlConnOfferedEv 8889001	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = Unknown, CurrentCalledAddress = 8889001, ModifiedCallingAddress = Unknown, ModifiedCalledAddress = 8889001, No LastRedirectedPartyAddress

Action	Events	Call info
irvCTIRD3's Active remote destination of 916267829523 answers the call.	GC1: ConnConnectedEvent 8889001 GC1: CallCtlConnEstablishedEv 8889001 GC1: TermConnCreatedEvent irvCTIRD3 GC1: TermConnActiveEvent irvCTIRD3 GC1: CallCtlTermConnTalkingEv irvCTIRD3 GC1: ConnCreatedEvent 8882000 GC1: ConnInprogressEvent 8882000 GC1: CallCtlConnOfferedEv 8882000 GC1: ConnAlertingEvent 8882000 GC1: CallCtlConnAlertingEv 8882000 GC1: TermConnCreatedEvent irvCTIPort2 GC1: TermConnRingingEvent irvCTIPort2 GC1: CallCtlTermConnRingingEv irvCTIPort2	CallingAddress = Unknown, CalledAddress 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress
Answer() the call on 8882000 terminal connection.	GC1: ConnConnectedEvent 8882000 GC1: CallCtlConnEstablishedEv 8882000 GC1: TermConnActiveEvent irvCTIPort2 GC1: CallCtlTermConnTalkingEv irvCTIPort2	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress
Disconnect() the call on 8889001 connection.	GC1: TermConnDroppedEv irvCTIRD3 GC1: CallCtlTermConnDroppedEv irvCTIRD3 GC1: ConnDisconnectedEvent 8889001 GC1: CallCtlConnDisconnectedEv 8889001 GC1: TermConnDroppedEv irvCTIPort2 GC1: CallCtlTermConnDroppedEv irvCTIPort2 GC1: ConnDisconnectedEvent 8882000 GC1: CallCtlConnDisconnectedEv 8882000 GC1: CallInvalidEvent GC1: CallObservationEndedEv	CallingAddress = Unknown, CalledAddress = 8889001, CurrentCallingAddress = 8889001, CurrentCalledAddress = 8882000, ModifiedCallingAddress = 8889001, ModifiedCalledAddress = 8882000, No LastRedirectedPartyAddress

CTI RD Call Forward

Table 1: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2, E - Enterprise Line

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observer on A, B and E	ProvInServiceEv	
A calls B	CallActiveEv on A, B, E ConnCreatedEv on A, B, E ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, E TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnCreatedEv on B ConnInProgressEv on B, E CallCtlConnOfferedEv on B, E ConnAlertingEv on A, B, E CallCtlConnAlertingEv on A, B, E TermConnCreated on Term B TermConnRingingEv on B, E CallCtlTermConnRingingEv on B, E	All RDD's will ring
C answers the call	**At Step 3:** ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	

Action	Events	Call Info
Disconnects the call	At Step 4: TermConnDroppedEv on A, B, E CallCtlTermConnDroppedEv on A, B, E ConnDisconnectedEv on A, B, E CallCtlConnDisconnectedEv on A, B, E CallInvalidEv CallObservationEndedEv on A, B, E	

Table 2: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected "is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Opens only A and no events for B	ProvInServiceEv	
GC1: A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B, ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B, ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will disconnect with message USER_BUSY	ConnFailedEv for A	USER_BUSY on Shared enterprise line

Table 3: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A- IP Phone, B - CTI-RD. VoiceMail Is Configured

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
GC1: A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnInProgressEv on VoiceMail of B CallCtlConnOfferedEv on VoiceMail of B ConnAlertingEv on VoiceMail of B CallCtlConnAlertingEv on VoiceMail of B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 4: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A - IP Phone, B - CTI-RD. VoiceMail Is Configured for B

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Only A is observed	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, C CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 5: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A IP Phone, B CTI-RD, C RDD1, D RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Applications adds C as the active remote destination on B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on CallCtlTermConnRingingEv on B	
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B CallCtlTermConnDroppedEv on A, B ConnDisconnectedEv on A, B CallCtlConnDisconnectedEv on A, B CallObservationEndedEv on A, B	

Table 6: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A IP Phone, B CTI-RD, C RDD1, D RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Applications adds C as the active remote destination on B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B CallCtlTermConnDroppedEv on A, B ConnDisconnectedEv on A, B CallCtlConnDisconnectedEv on A, B CallObservationEndedEv on A, B	

Table 7: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2, E - Enterprise Line

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observer on A, B and E	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, E ConnCreatedEv on A, B, E ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, E TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, E CallCtlConnAlertingEv on A, B, E ConnInProgressEv on B, E CallCtlConnOfferedEv on B, E TermConnRingingEv on B, E CallCtlTermConnRingingEv on B, E	All RDD will ring
C answers the call	ConnConnectedEv on B CallCtlConnEstablishedEv on B TermConnActiveEvent on B CallCtlTermConnTalkingEv on B	
C Disconnects the call	TermConnDroppedEv on A, B, E CallCtlTermConnDroppedEv on A, B, E ConnDisconnectedEv on A, B, E CallCtlConnDisconnectedEv on A, B, E CallObservationEndedEv on A, B, E	

Table 8: Phone A Calls CTIRD When CTI Remote Device Is Observed, Active RD Is Not Set and "Route Calls to All Remote Destinations When Client Is Not Connected "is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B, ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B, TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, CallCtlConnAlertingEv on A, B, ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	All RDD will ring
call will disconnect with message USER_BUSY		USER_BUSY on Shared enterprise line

Table 9: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Enabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B, ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

Table 10: Phone A Calls CTIRD When CTI Remote Device Is Observed, Remote Destination Is Not Configured and "Route Calls to All Remote Destinations When Client Is Not Connected" Is Disabled; A - IP Phone, B - CTI-RD, C - RDD1, D - RDD2

Action	Events	Call Info
User1 Opens Provider and adds a provider observer Add Call Observers on A and B	ProvInServiceEv	

Action	Events	Call Info
A calls B	CallActiveEv on A, B ConnCreatedEv on A, B ConnConnectedEv on A CallCtlConnInitiatedEv on A TermConnCreatedEv on A, B TermConnActiveEvent on A CallCtlTermConnTalkingEv on A CallCtlConnDialingEv on A CallCtlConnEstablishedEv on A ConnAlertingEv on A, B, C CallCtlConnAlertingEv on A, B ConnInProgressEv on B CallCtlConnOfferedEv on B TermConnRingingEv on B CallCtlTermConnRingingEv on B	
Call will Route to Voice mail number		Call will route to voice mail number

CTI Video Support

Use cases related to CTI Video Support feature are mentioned below:

Scenario 1:

Phone A is video capable, telepresence capable, with 1 screen and a camera, and in registered state. User1 has phone A in the control list. User invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() before opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
User1 invokes CiscoTerminal.i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() = TELEPRESENCEINTEROP_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 2

Phone A is not video capable, not telepresence capable with 0 screens. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() before opening device

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 3

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() after opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 4

Phone A is video not capable, not telepresence capable with 0 screens. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability() after opening the device.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 5

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 does not have phone A in the control list. User1 has Super provider capabilities.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 aquires phone A using prov.createTerminal("phoneA")	CiscoTermCreatedEv TermA CiscoAddrCreatedEv A	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_ENABLED
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 1

Scenario 6

Phone A is not video capable, not telepresence capable and has 0 screens. User1 does not have phone A in the control list. User1 has Super provider capabilities

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 aquires phone A using prov.createTerminal("phoneA")	CiscoTermCreatedEv TermA CiscoAddrCreatedEv A	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () = TELEPRESENCEINTEROP_NONE
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		termA. getCiscoMultiMediaCapabilityInfo(). getScreenCount () = 0

Scenario 7

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registeres it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Action	Events	Call info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo () on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount () on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario 8

Basic Video call: Phone A is video enabled, telepresence enabled with 1 screen. Phone B is video disabled , telepresence disabled with 0 screens. Both the phones are in the control list of User1.

Action	Events	Call info
User Opens provider and adds observer	ProvInServiceEv	
User adds terminal observers on Phone A and Phone B	CiscoTermInServiceEv TermA CiscotermInServiceEv TermB	
User adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call info
User makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1		The API returns 1, indicating video capable device(VIDEO_ENABLED) for TermA(far-end party).
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1		The API returns 1, indicating telepresence capable device(TELEPRESENCEINTEROP_ENABLED) for TermA(far-end party).
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo. getScreenCount() on GC1		The API returns 1, indicating device has 1 screen, for TermA(far-end party).
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1		The API returns 0, indicating video capable device(NONE) for Term B
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1		The API returns 1, indicating device is not telepresence capable (TELEPRESENCEINTEROP_NONE) for TermA(far-end party).
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo .getScreenCount() on GC1		The API returns 1, indicating device has 0 screens, for TermA(far-end party).

Scenario 9

Phone A is video disabled (in CUCM Admin Phone page, the Video Capabilities field is 'Disabled') , but the device has an an external camera (USB or CUVA) plugged in. Phone A is in registered state.

Action	Events	Call info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE

Action	Events	Call info
In Device Configuration CUCM Admin pages- Video Capabilities field is changed to'Enabled'	CiscoProvTerminalMultiMedia CapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
The external camera is removed, or the Cisco Camera field in CUCM Admin Phone page is 'Disabled'	No event is delivered, as the device is still a video capable device as it can receive video	
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = VIDEO_ENABLED
In Device Configuration CUCM Admin pages- Video Capabilities field is changed to'Disabled'	CiscoProvTerminalMultiMedia CapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = NONE

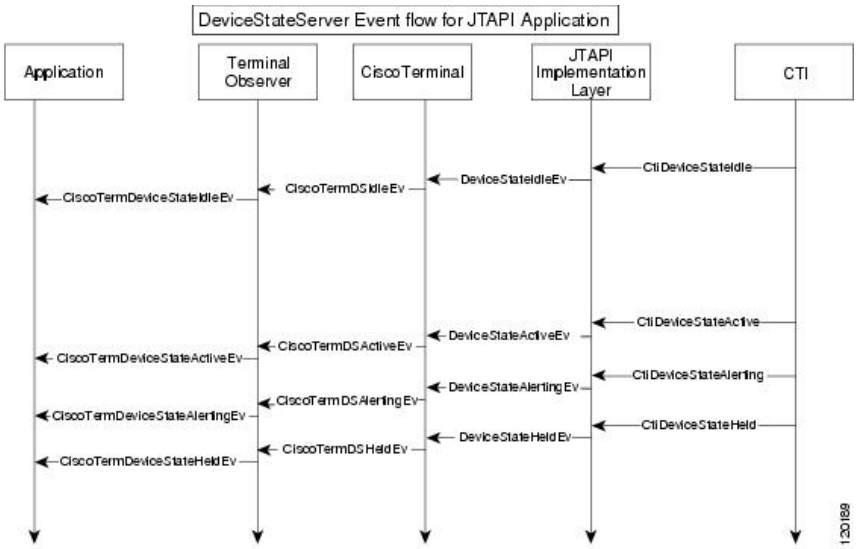
Device and Line Restriction

S.No	Scenario	Events
1	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list. T1 and A3 is added into the restricted list. Application opens the provider	
	Application queries for is Restricted on T1, T2, T3	CiscoTerminal.isRestricted() returns true for T1 and false for T2 and T3
	Application queries for is Restricted on Address A1, A2, A3	CiscoAddress.isRestricted() returns true for A1, A3, false for A2.
		CiscoAddress.getRestrictedAddrTerminals() on A1, A3 returns T1, T3 respectively, returns null for A2.
	Application tries to addObserver and addCallObserver on T1, T2, T3, A1, A2, A3	addObserver and addCallObserver fails for T1, A1, A3. For T3 observer is added, but no events are received on A3. For A2, application will be able to add observers successfully and events will be received
2	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.	

S.No	Scenario	Events
	Application opens the provider and adds observer on all terminals and addresses.	CiscoTermRestrictedEv for T1 CiscoAddrRestrictedEv for L1 CiscoAddrRestrictedEv for A2 sent to providerObserver.
	T1 and A2 are added to the restrictedlist.	CiscoTermOutOfServiceEv for T1 CiscoAddrOutOfServiceEv for L1 CiscoAddrOutOfServiceEv for A2
	T1 and L2 are removed from restricted list	CiscoTermActivatedEv for T1 and CiscoAddrActivatedEv for A1 CiscoAddrActivatedEv for A2 sent to providerObserver. CiscoTermInServiceEv for T1 and CiscoAddrInServiceEv for A1 CiscoAddrInServiceEv for A2 sent to terminal and address observers.
3	Application has Devices T1, T2, T3 whose lines are A1, A1, A2 in the control list. A1 is the shared line on T1 and T2 Application opens provider and adds observer on all terminals/addresses	
	T1 is added into the restricted list.	Application will see CiscoTermRestrictedEv for T1 and CiscoAddrRestrictedOnTerminalEv which contains getAddress is L1 and getTerminal as T1. Application will also see CiscoTermOutOfServiceEv for T1 and CiscoAddrOutOfService for A1/T1
	T1 is removed from the restricted list	CiscoTermActivatedEv for T1 CiscoAddrActivatedEv for L1 CiscoTermInServiceEv for T1 CiscoAddrInServiceEv for A1/T1
4	Application has Devices T1, T2, T3 whose lines are A1, A1, A1 in the control list. A1 is the shared line on T1, T2 and T3 Application opens the provider and adds observer on all terminals and addresses	
	A1 on T1 is added to the restricted list	CiscoAddrRestrictedOnTerminalEv for A1/T1 CiscoAddrOutOfServiceEv for A1/T1
	A1 on T2 is added to therestricted list	CiscoAddrRestrictedOnTerminalEv for A1/T2 CiscoAddrOutOfServiceEv for A1/T2
	A1 on T3 is added to therestricted list	CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1/T3

S.No	Scenario	Events
	A1 on T1 is removed from the restricted list	CiscoAddrActivatedOnTerminalEv for A1/T1 CiscoAddrInServiceEv for A1/T1
	A1 on T2 is removed from the restricted list	CiscoAddrActivatedOnTerminalEv for A1/T2 CiscoAddrInServiceEv for A1/T2
	A1 on T3 is removed from the restricted list	CiscoAddrActivatedEv for A1 CiscoAddrInServiceEv for A1/T3
5	Application has Devices T1, T2, T3 whose lines are A1, A2, A3 in the control list.	
	Application opens the provider and adds observer on all terminals and addresses. A1 is involved in a call with party X.	CiscoAddrRestrictedEv for A1 CiscoAddrOutOfServiceEv for A1
	A1 is added into the restricted list.	ConnDisconnectedEv CallCtlConnDisconnectedEv TermConnDroppedEv CallCtlConnDroppedEv CallInvalidEv

Device State Server



Do Not Disturb

Configuration: Application is observing terminal A and terminal B.

Scenario One

Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is enabled via setDNDChangedEvFilter. DND is enabled on the terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application adds terminal observer to terminal A. Filter is enabled via setDNDChangedEvFilter() in CiscoTermEvFilter. DND is enabled on the terminal through phone or admin page. Application invokes getDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to the application	N.A

Scenario Two

Application enables filter to receive events. Application adds Terminal observer to terminal A using Terminal.addObserver(). DND is enabled on the terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application enables filter to receive events. Application adds terminal observer to terminal A. DND is enabled on the device through phone or admin pages. Application invokes getDNDStatus() from CiscoTerminal.	NEW META EVENT_____META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to the application	N.A

Scenario Three

Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is disabled via setDNDChangedEvFilter() in CiscoTermEvFilter. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application adds Terminal observer to terminal A using Terminal.addObserver(). Filter is disabled via setDNDChangedEvFilter() in CiscoTermEvFilter. Application invokes getDNDStatus() from CiscoTerminal.	CiscoTermDNDStatusChangedEv is not delivered to application.	N.A

Scenario Four

Application does not add Terminal observer to terminal. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application does not add Terminal observer to terminal. Application invokes getDNDStatus() from CiscoTerminal.	InvalidStateException is thrown	N.A

Scenario Five

Application does not enable the filter to receive events. Application adds Terminal observer to terminal A. DND status is set to true through the phone or admin pages. Application now enables the filter to receive events. Application invokes getDNDStatus() from CiscoTerminal.

Action	Events	Call info
Application does not enable the filter to receive events.Application adds Terminal observer to terminal A. DND status is set to true through the phone or admin pages. Application now enables the filter to receive events Application invokes getDNDStatus() from CiscoTerminal.	CiscoTermDNDStatusChangedEv is not delivered to application. NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true is returned to application	N.A

Scenario Six

Application sets DND status to false by invoking the setDNDStatus() interface on CiscoTerminal.

Action	Events	Call info
Application invokes setDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A

Scenario Seven

Application 1 and Application 2 are observing terminal a, and both the applications have enabled the filter to receive events. Application 1 sets DND status to false on Terminal A. Application 2 is observing Terminal A.

Action	Events	Call info
Application invokes setDNDStatus() from CiscoTerminal.	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A

Scenario Eight

DND Type is RingerOff and CFNA is not set. Terminal B calls Terminal A. Call is presented to A and call is not answered.

Action	Events	Call info
Application invokes redirect() API with feature priority set to 3 from CiscoCall.	Call is presented to the device, irrespective of the DND settings on the device. CER call overrides DND setting.	N.A
Application invokes selectRoute() API with feature priority set to 3 from CiscoRouteSession.	Call is presented to the device, irrespective of the DND settings on the device. CER call overrides DND setting.	N.A

Scenario Nine

Action	Events	Call info
DND Type is RingerOff and CFNA is not set. Terminal B calls Terminal A .Call is presented to A and call is not answered.	ConnFailedEv Cause: CAUSE_NO ANSWER	N.A

Scenario Ten

DND Type is CallReject and CFB is not set. Terminal B calls Terminal A. Call is not presented to A.

Action	Events	Call info
DND Type is CallReject and CFB is not set. Terminal B calls Terminal A. Call is not presented to A	ConnFailedEv Cause: CAUSE_USER BUSY	N.A

Scenario Eleven

DND is enabled on the terminal A. Terminal A comes IN_SERVICE. Application invokes getDNDStatus() on CiscoTerm in ServiceEv.

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE.	CiscoTermInServiceEv Cause: CAUSE_NORMAL DND Status = true	N.A

Scenario Twelve

DND is enabled on terminal A. Terminal A comes IN_SERVICE. Application invokes setDNDStatus(). DB failure happens after the setDNDStatus() request is sent.

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE. Application invokes setDNDStatus(). DB failure follows and value is not updated in DB.	PlatformException is thrown “Could not meet post conditions of setDNDStatus()” No CiscoTermDNDStatusChangedEv is received.	N.A

Scenario Thirteen

DND is enabled on the terminalA. Terminal A comes IN_SERVICE, DND status is currently true in phone/admin. Application tries to set the same value i.e. invokes setDNDStatus(true).

Action	Events	Call info
DND is enabled on the terminal A. Terminal A comes IN_SERVICE.DND status is currently true in phone/admin.Application tries to set the same value i.e. invokes setDNDStatus(true).	InvalidStateException is caught: DND status with value true is already set No CiscoTermDNDStatusChangedEv is received.	N.A

DND-R

Scenario One

Application adds Terminal observer to terminal A using Terminal.addObserver (). DND-R is enabled on the terminal B via the Admin page or the Common profile page.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page. A issues Call.connect to B with the feature Priority = 1 (Normal)	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL ConnFailedEv B Cause:. Cause: CAUSE_USERBUSY.	N.A

Scenario Two

Application adds Terminal observer to terminal A using Terminal.addObserver (). DND-R is enabled on the Terminal B via the Admin page or the Common profile page.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page. A issues Call.connect to B with the feature Priority = 3 (Emergency)	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause:CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause:CAUSE_NORMAL ConnInProgressEv for B Cause:CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause:CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A Called: B

Scenario Three

DND-Call reject with CFB not set.

Action	Events	Call info
Application adds terminal observers to terminal A and B. DND-R is enabled on the terminal B through phone or admin page with no CFB Setting. Terminal A issues Call.connect to Terminal B.	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause:CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause:CAUSE_NORMAL ConnFailedEv B Cause:CAUSE_USERBUSY.	NA

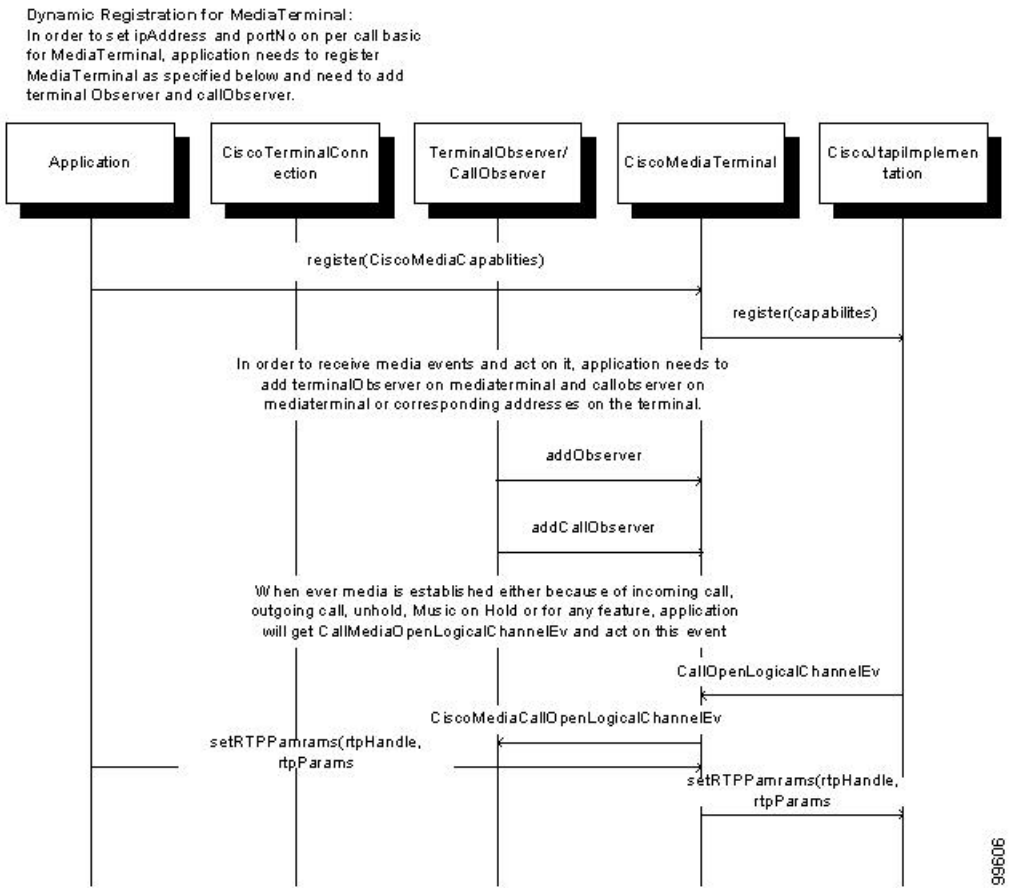
Scenario Four

DND – Call reject with CFB set to C.

Action	Events	Call info
Application is Observing Terminal A, B & C. DND-R is Enabled in Terminal B with CFB set to Terminal C. Terminal A issues Call.connect to Terminal B. Call is not Presented on Terminal B and is Forwarded to Terminal C.	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for C cause: REDIRECTED CALL ConnInProgressEv for C Cause: REDIRECTED CALL CallCtlConnOfferedEv for C Cause: REDIRECTED CALL ConnAlertingEv for C Cause REDIRECTED CALL CallCtlConnAlertingEv for C Cause: REDIRECTED CALL TermConnCreatedEv for C Cause: REDIRECTED CALL TermConnRingingEv for C Cause: REDIRECTED CALL CallCtlTermConnTalkingEv Cause: CAUSE_REDIRECTED	Calling: A Called: C LastRedirectedParty: B

Dynamic CTIPort Registration Per Call

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



E911 Teleworker

SelectRoute Method

Use Case	Events	Call info
No changes in use case with selectRoute Method	No changes in the events	No changes in the callinfo

Redirect Method

Use Case	Events	Call info
No changes in use case with redirect Method	No changes in the events	No changes in the callinfo

Encryption Enhancement

Table 11: Service Parameter "Require Public Key Encryption" Is Set to "False". Application Is Using a Pre 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 12: Service Parameter "Require Public Key Encryption" Is Set to "True". Application Is Using a Pre 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider	PlatformException	getErrorCode() = CiscoJtapiException. INCOMPATIBLE_PROTOCOL_ VERSION

Table 13: Service Parameter "Require Public Key Encryption" Is Set to "False". Application Is Using a 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 14: Service Parameter "Require Public Key Encryption" Is Set to "True". Application Is Using a 10.x CiscoJTAPI Version

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	

Table 15: SP Is Set to "True". Application Is Using 10.x CiscoJTAPI Lib. Application Has Provided Pub and Sub Ctimanager IP in the ProviderString

Action	Result	Info
Application opens a provider and adds a provider observer	ProvInServiceEv	
CTIManager on the server to which app is connected goes down	ProvOutOfService	

End to End Call Tracing

Actions	Events	Call info
1.a) Both A and B are in user's control list. Basic Call A calls B; App is observing only B	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A … … GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlTermConnTalkingEv for TB	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID1 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(TB) returns ID1 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(term) will throw InvalidStateException
1.b) Only B is in User's control list Basic Call A calls B; App is observing only B	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A …… …… GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlTermConnTalkingEv for TB	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID1 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(TB) returns ID1 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(term) will throw PrivilegeViolationException

Actions	Events	Call info
2.a) Redirect in offering state Scenario All Observed A calls B; B redirects the call to C in offering state C Accepts the call C answers the call	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlConnEstablishedEv for C GC1: CallCtlTermConnTalkingEv for TC	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID2 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID3 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID4

Actions	Events	Call info
2.b) Redirect in connected state Scenario All Observed A calls B B answers B redirects the call to C in connected state	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID6 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID7 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID8

Actions	Events	Call info
3. Redirect Scenario Only B & C are Observed A calls B; B answers B redirects the call to C in connected state	GC1: CallActiveEv GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: TermConnCreatedEv for TB GC1: TermConnActiveEvent for TB GC1: CallCtlConnEstablishedEv for A GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnTalkingEv for TB GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnOfferedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B	((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID10 ((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) throws InvalidStateException ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID11

Actions	Events	Call info
4. Conference Scenario; All observed GC1: A calls B; B answers GC2: B calls C; C answers GC1.conference(GC2)		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID12 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID13 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID14 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID15 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID16

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnHeldEv for TB GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for TB GC2: TermConnActiveEvent for TB GC2: CallCtlTermConnTalkingEv for TB GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnOfferedEv for C GC2: TermConnCreatedEv for TC GC2: ConnConnectedEv for B GC2: CallCtlConnEstablishedEv for B GC2: ConnConnectedEv for C GC1: CiscoConferenceStartEv GC1: CiscoCallFeatureCancelledEv GC2: CiscoCallChangedEv	

Actions	Events	Call info
	GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoConferenceEndEv	

Actions	Events	Call info
5. Transfer Scenario; All observed GC1: A calls B; B answers GC2: B calls C; C answers GC1.transfer(GC2)		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID19 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID20 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID21 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID22 ((CiscoConnection)(ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID23

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: CallCtlTermConnHeldEv for TB GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for TB GC2: TermConnActiveEvent for TB GC2: CallCtlTermConnTalkingEv for TB GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnOfferedEv for C GC2: TermConnCreatedEv for TC GC2: ConnConnectedEv for B GC2: CallCtlConnEstablishedEv for B GC2: ConnConnectedEv for C GC2: CallCtlConnEstablishedEv for C GC1: CiscoTransferStartEv GC2: CiscoCallChangedEv	

Actions	Events	Call info
	GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	

Actions	Events	Call info
6. Shared Line Scenario; All Observed; DN B is present on T1, T2 A calls B; B(T1) Answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: TermConnCreatedEv for T1B GC1: TermConnRingingEv for T1B GC1: CallCtlTermConnRingingEv for T1B GC1: TermConnCreatedEv for T2B GC1: TermConnRingingEv for T2B GC1: CallCtlTermConnRingingEv for T2B GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for T1B GC1: CallCtlTermConnTalkingEv for T1B GC1: TermConnPassiveEv for T2B GC1: CallCtlTermConnBridgedEv	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(TA) returns ID25 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID26 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(T1) returns ID26 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(T2) returns ID26 (Connection of B).getUniqueID(null) returns returns ID26 (Connection of B).getUniqueID(T1) returns returns ID26 (Connection of B).getUniqueID(T2) returns returns ID26

Actions	Events	Call info
7. Shared Line Barge Scenario; All Observed; DN B is present on T1, T2 A calls B; B(T1) Answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: TermConnCreatedEv for T1B GC1: TermConnRingingEv for T1B GC1: CallCtlTermConnRingingEv for T1B GC1: TermConnCreatedEv for T2B GC1: TermConnRingingEv for T2B GC1: CallCtlTermConnRingingEv for T2B GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for T1B GC1: CallCtlTermConnTalkingEv for T1B GC1: TermConnPassiveEv for T2B GC1: CallCtlTermConnBridgedEv	((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(TA) returns ID27 (Connection of B).getUniqueID(null) returns returns ID28 (Connection of B).getUniqueID(T1) returns returns ID28 (Connection of B).getUniqueID(T2) returns returns ID28

Actions	Events	Call info
B(T2) Presses Barge	GC2: CallActiveEv GC2: ConnCreatedEv for B GC2: ConnConnectedEv for B GC2: CallCtlConnInitiatedEv for B GC2: TermConnCreatedEv for T2B GC2: TermConnCreatedEv for T1B GC2: CiscoCallChangedEv GC1: TermConnActiveEv for T2B GC1: CallCtlTermConnTalkingEv for T2B GC2: TermConnDroppedEv for T2B GC2: CallCtlTermConnDroppedEv for T2B GC2: TermConnDroppedEv for T2B GC2: CallCtlTermConnDroppedEv for T2B GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC2: CallInvalidEv GC2: CallObservationEndedEv	((CiscoConnection)(GC2: ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID29 (GC1: Connection of B).getUniqueID(T1) returns returns ID28 (GC1: Connection of B).getUniqueID(T2) returns returns ID30 (GC1: Connection of B).getUniqueID(null) is unpredictible (can be either of above two)

Actions	Events	Call info
8. Park/Unpark Scenario (All Observed) A calls B; B answers B does PARK C Unparks		((CiscoConnection)(ConnCreatedEv for A).getConnection()).getUniqueID(null) returns ID31 ((CiscoConnection)(ConnCreatedEv for B).getConnection()).getUniqueID(null) returns ID32 ((CiscoConnection)(GC1: ConnCreatedEv for ParkDN).getConnection()).getUniqueID(null) throws PrivilegeVoilationException ((CiscoConnection)(GC2: ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID34 ((CiscoConnection)(GC1: ConnCreatedEv for C).getConnection()).getUniqueID(null) returns ID35

Actions	Events	Call info
	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for A GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: ConnCreatedEv for B GC1: ConnConnectedEv for B GC1: CallCtlConnOfferedEv for B GC1: TermConnCreatedEv for TB GC1: ConnConnectedEv for A GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for ParkDN GC1: CallCtlConnQueuedEv for ParkDN GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC2: CallActiveEv GC2: ConnCreatedEv for C GC2: ConnConnectedEv for C GC2: CallCtlConnInitiatedEv for C GC2: TermConnCreatedEv for TC GC2: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: TermConnCreatedEv for TC GC1: ConnDisconnectedEv for ParkDN GC1: CallCtlConnDisconnectedEv for ParkDN GC2: TermConnDroppedEv for TC	

Actions	Events	Call info
	GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC2: CallInvalidEv GC2: CallObservationEndedEv	
9. Conference Chaining Scenario; All Observed GC1: A calls B; B answers and adds C to conference GC3: C calls D; D answers and adds E to conference App sends GC1.conference(GC3) to chain two conferences	GC1 and GC3 are created as normal GC1 has connections for A, B and C(Held) GC3 has connections for C(Talking), D and E GC3: ConnCreatedEvent for cBridge-GC1 GC3: CiscoConferenceChainAddedEv GC3: ConnConnectedEvent for cBridge-GC1 GC3: CallCtlConnEstablishedEv for cBridge-GC1 GC3: TermConnDroppedEv for TC GC3: CallCtlTermConnDroppedEv for TC GC3: ConnDisconnectedEvent for C GC3: CallCtlConnDisconnectedEv C GC1: CallCtlTermConnTalkingEv for TC GC1: ConnCreatedEvent for cBridge-GC3 GC1: CiscoConferenceChainAddedEv GC1: ConnConnectedEvent for cBridge-GC3 GC3: CallCtlConnEstablishedEv for cBridge-GC3	(Connection for A).getUniqueID(null) returns ID37 (Connection for B).getUniqueID(null) returns ID38 (GC1: Connection for C).getUniqueID(null) returns ID39 (Connection for C).getUniqueID(null) returns ID40 (Connection for D).getUniqueID(null) returns ID41 (GC3: Connection for E).getUniqueID(null) returns ID42 ((CiscoConnection)(GC3: ConnCreatedEv for cBridge-GC1).getConnection()).getUniqueID(null) throws PrivilegeVoilationException ((CiscoConnection)(GC1: ConnCreatedEv for cBridge-GC3).getConnection()).getUniqueID(null) throws PrivilegeVoilationException (Connection for A).getUniqueID(null) returns ID37 (Connection for B).getUniqueID(null) returns ID38 (Connection for C).getUniqueID(null) returns ID39 (Connection for D).getUniqueID(null) returns ID41 (Connection for E).getUniqueID(null) returns ID42

Actions	Events	Call info
Application sends E.disconnect()	GC3: TermConnDroppedEv for TE GC3: CallCtlTermConnDroppedEv for TE GC3: ConnDisconnectedEvent for E GC3: CallCtlConnDisconnectedEv E GC1: ConnDisconnectedEvent for cBridge-GC3 GC1: CiscoConferenceChainRemovedEv GC1: CallCtlConnDisconnectedEv cBridge-GC3 GC3: CiscoCallChangedEv GC1: ConnCreatedEvent for D GC1: ConnConnectedEvent for D GC1: CallCtlConnEstablishedEv for D GC1: TermConnCreatedEvent for TD GC1: TermConnActiveEvent for TD GC1: CallCtlTermConnTalkingEv for TD GC3: ConnDisconnectedEvent for cBridge-GC1 GC3: CiscoConferenceChainRemovedEv GC3: CallCtlConnDisconnectedEv cBridge-GC31 GC3: TermConnDroppedEv for TD GC3: CallCtlTermConnDroppedEv for TD GC3: ConnDisconnectedEvent for D GC3: CallCtlConnDisconnectedEv D GC3: CallInvalidEvent GC3: CallObservationEndedEv	((CiscoConnection)(GC1: ConnCreatedEv for D).getConnection()).getUniqueID(null) returns ID43

Hunt Log Status for Phone Devices

In the following use cases A, B, C, and D are IP phones where A and B are a part of line group which is configured to hunt pilot HP. A is the first hunt member and it is logged out of the hunt group and B is the second hunt member and it is logged in to the hunt group on hunt pilot. For the following use cases the CiscoTermEvFilter. setHuntLogStatusChangedEvFilter() is set to true.

Call To Hunt Pilot where device is logged into hunt group

Action	Events	Call information
On A, huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus).	CiscoTermHuntLogStatusChangedEv on A	Ev.getHutLogStatus() = CiscoTerminal.DEVICE_HUNT_LOGGED_IN
C calls Hunt pilot HP.	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 CiscoHuntConnCreatedEv HP GC1 ConnInProgressEv HP GC1 CallCtlConnOfferedEv HP	CurrentCallingParty = C CurrentCalledParty = A
HP offers the call to A as it is the first hunt member and A starts ringing.	GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAltertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TermA GC1 TermConnRingingEv TermA GC1 CallCtlTermConnRingingEv TermA	
A answers the call	GC1 ConnConnectedEv HP1 GC1 CallCtlConnEstablishedEv HP1 GC1 ConnConnectedEv A GC1 CallCtlConnEstablishedEv A GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA	CurrentCallingParty = C CurrentCalledParty = A

Call To Hunt Pilot where device is logged out of the hunt group

Action	Events	Call information
On A, huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_OUT using the method setHuntLogStatus(int huntLogStatus) for the terminal to log in to the huntgroup.	CiscoTermHuntLogStatusChangedEv on A	Ev.getHuntLogStatus() = CiscoTerminal.DEVICE_HUNT_LOGGED_OUT
C calls Hunt pilot HP.	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC	CurrentCallingParty = C CurrentCalledParty = B
HP offers the call to B as A which is the first hunt member logged out of the hunt group and B is the second hunt member. B starts ringing.	GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 CiscoHuntConnCreatedEv HP GC1 ConnInProgressEv HP GC1 CallCtlConnOfferedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	
B answers the call	GC1 ConnConnectedEv HP1 GC1 CallCtlConnEstablishedEv HP1 GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB	CurrentCallingParty = C CurrentCalledParty = B

Updating the value of huntlogstatus on Unsupported device(Route Point/Spark Remote device/CTI Remote Device)

Action	Events	Call information
huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus) for the route point to log in to the huntgroup.		com.cisco.jtapi.MethodNotSupportedException:Operation not allowed

Updating the value of huntlogstatus on the terminal which is out of service

Action	Events	Call information
huntLogStatus is set to CiscoTerminal. DEVICE_HUNT_LOGGED_IN using the method setHuntLogStatus(int huntLogStatus) on D which is not in service		com.cisco.jtapi.InvalidStateException

Energywise Deep Sleep Mode

Scenario 1

JTAPI reports new reason“ENERGYWISE_POWER_SAVE_PLUS”in CiscoProvTerminalUnRegisteredEv and cause“CAUSE_ENERGYWISE_POWER_SAVE_PLUS”in CiscoTermOutOfServiceEv and CiscoAddrOutOfServiceEv to the application when a terminal/address unregisters from Cisco Unified CM due to deep sleep time.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of 'A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	
Terminal 'A' enters Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

Scenario 2

Terminal gets unregistered due to Deep Sleep mode and the user tries to manually register the terminal during the Deep Sleep time.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of 'A"	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	
Terminal 'A' goes to Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
A user tries to register the phone with Cisco Unified CM during deep sleep mode.		Cisco Unified IP 7900 Series phones do not re-register with the Cisco Unified CM during the Deep Sleep time. This is a limitation of the phone. Cisco Unified IP 9900 and 6900 Series phones register back with the Cisco Unified CM by pressing the select key on the phone.
Phone registers with the Cisco Unified CM after the Deep Sleep time expires.	[Term A] CiscoProvTerminalRegisteredEv [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv	

Scenario 3

Shared line scenario. Two devices A (Cisco Unified IP Phones 7900 Series phone) and A' (CTI Port) are configured with the same line. Deep Seep mode is enabled on device A

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	

Description	Events	Information
Terminal A goes to Deep Sleep mode and gets unregistered. (Terminal A' remains in registered state.)	[Term A] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

Scenario 4

Shared line scenario. Two devices A and A' (both are Cisco Unified IP Phones 7900 Series phones) that have are configured with the same line. Deep Sleep mode is enabled on A. Another device B calls the shared line after device A enters to the Deep Sleep mode.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	
Terminal A goes to deep sleep mode and gets unregistered. (Terminal A' remains in registered state.)	[Term A] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv	CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER

Description	Events	Information
Another terminal B calls to the shared line DN.	GC1 CallActiveEv GC1 ConnCreatedEv [Addr B] GC1 ConnConnectedEv [Addr B] GC1 CallCtlConnInitiatedEv [Addr B] - GC1 ConnCreatedEv [Addr A'] GC1 ConnInProgressEv [Addr A'] GC1 CallCtlConnOfferedEv [Addr A'] - GC1 ConnConnectedEv [Addr A'] GC1 CallCtlConnEstablishedEv [Addr A'] GC1 TermConnActiveEv [Term A'] GC1 CallCtlTermConnTalkingEv [Term A'	

Scenario 5

Shared line scenario. Two device A (Cisco Unified IP Phones Series 9900/6900 phone) and A' (Cisco Unified IP Phones Series 9900/6900 phone) are configured with the same line. Deep Sleep mode is enabled on both devices.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and A'	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term A'] CiscoTermInServiceEv [Addr A'] CiscoAddrInServiceEv	

Description	Events	Information
Terminal A and A' enters Deep Sleep mode and gets unregistered	[Term A] CiscoProvTerminalUnRegisteredEv [Term A'] CiscoProvTerminalUnRegisteredEv [Term A] CiscoTermOutOfServiceEv [Term A'] CiscoTermOutOfServiceEv [Addr A] CiscoAddrOutOfServiceEv [Addr A'] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
Deep Sleep mode power off time has expired and A and A' reregister to the Cisco Unified CM	Term A] CiscoProvTerminalRegisteredEv [Term A'] CiscoProvTerminalRegisteredEv [Term A] CiscoTermInServiceEv [Term A'] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Addr A'] CiscoAddrInServiceEv	

Scenario 6

Basic call scenario. Two devices A (CTI port) and B (Cisco Unified IP Phones 7900 Series phone) are configured on a Cisco Unified CM and Deep Sleep mode is enabled on B with power off time configured for 6:00 PM. A calls B at 5:55 pm and the call continues until 6:10 pm. The idle timer is set for 10 minutes.

Description	Events	Information
Application opens the provider and adds observer on provider, terminal and address of A and B	ProvInServiceEv P1 [Term A] CiscoTermInServiceEv [Addr A] CiscoAddrInServiceEv [Term B] CiscoTermInServiceEv [Addr B] CiscoAddrInServiceEv	

Description	Events	Information
Terminal B calls A at 5:55 pm. A answers the call and goes to connected state.	GC1 CallActiveEv GC1 ConnCreatedEv [Addr B] GC1 ConnConnectedEv [Addr B] GC1 CallCtlConnInitiatedEv [Addr B] - GC1 ConnCreatedEv [Addr A] GC1 ConnInProgressEv [Addr A] GC1 CallCtlConnOfferedEv [Addr A] - GC1 ConnConnectedEv [Addr A] GC1 CallCtlConnEstablishedEv [Addr A] GC1 TermConnActiveEv [Term A] GC1 CallCtlTermConnTalkingEv [Term A]	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS
At 6:00 pm Deep Sleep time is enabled but the call does not get dropped and remains active.		
The user disconnects the call from the phone at 6:10 PM and the idle timer (set for 10 minutes) starts.	GC1 TermConnDroppedEv [Term B] GC1 CallCtlTermConnDroppedEv [Term B] GC1 ConnDisconnectedEv [Addr B] GC1 CallCtlConnDisconnectedEv [Addr B] GC1 CallInvalidEv	
There is no action on phone A for the next 10 minutes. So at 6:20 pm, after the idle timer has expired, the terminal enters Deep Sleep mode and unregisters from the Cisco Unified CM.	[Term B] CiscoProvTerminalUnRegisteredEv [Term B] CiscoTermOutOfServiceEv [Addr B] CiscoAddrOutOfServiceEv	CiscoProvTerminalUnRegisteredEv.getReason() = CiscoProvTerminalUnRegisteredEv. ENERGYWISE_POWER_ SAVE_PLUS CiscoTermOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS CiscoAddrOutOfServiceEv.getCause() = CiscoOutOfServiceEv. CAUSE_ENERGYWISE_POWER_ SAVE_PLUS

External Call Control

You should assume that all devices in the following use cases are obsereved, unless explicilty stated otherwise in the use case description.

The first few use cases go through the full event series for the basic call setup. After the first three or four, the use cases leave this part out, as it is standard for most of the use cases. If you do not see the basic call event series at the beginning of a use case, you can assume that it was intended to have happened successfully before the first event in the use case.

The last column in the use cases, that specifies the call info for a various stage of the use case, will initially have the full method invocation to retrieve the call information, for example CiscoCall.getModifiedCallingParty(). After the first three or four uses cases, only the method name is specified, such as .getModifiedCallingParty(). You can assume that this is to be prefixed with CiscoCall unless explicitly stated otherwise, such as for the CiscoCallChangeEvs.

Use Cases for BasicCall

Basic Call Initiated From JTAPI / Phone

Configuration

Phone A, B are in cluster devices.

Procedure:

Application invokes connect() at A to call B, or physical phone for A dials the number for B.

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A	CiscoCall.getCurrentCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getCurrntCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall getCurrentCalledTerminal() = null
Connection of B created B starts ringing B Answers	GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B, CiscoCall.getCalledAddress() = B, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B

Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Called Party

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where called party get transformed to B1.

Procedure:

Application invokes connect() at A to call B.

Actions	Events	Call info
A initiates call to B Connection of A created,	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null

Actions	Events	Call info
Connection of B1 created B1 starts ringing B1 Answers	GC1-ConnCreatedEvent-B1 GC1-ConnInprogressEvent-B1 GC1-CallCtlConnOfferedEv-B1 GC1-ConnAlertingEvent-B1 GC1-CallCtlConnAlertingEv-B1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B1 GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getCurrentCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getCurrentCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Calling Party

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where calling party gets transformed to A1.

Procedure:

Application invokes connect() at A to call B.

Action	Events	Call info
A initiates call to B Connection of A created, Connection of B created B starts ringin	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall getCallingAddress() = A, CiscoCall getCurrentCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B, CiscoCall getCalledAddress() = B, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null
B Answers	GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B, CiscoCall.getCalledAddress() = B, CiscoCall.getCurrentCallingTerminal() = Terminal of A CiscoCall.getCurrentCalledTerminal() = Terminal of B

Basic Call Initiated From JTAPI to the DN with Translation Pattern Configured to Transform Both Calling and Called Parties

Configuration

Phone A, B are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B

Action	Events	Call info
A initiates call to B Connection of A created, called party info set Connection of B1 created B1 starts ringing A gets CallStateChg For Ringback	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-B	CiscoCall.getModifiedCallingAddress() = A, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = “”, CiscoCall.getCalledAddress() = “”, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall getCallingAddress() = A, CiscoCall getModifiedCalledAddress() = B1, CiscoCall getCalledAddress() = B1, CiscoCall getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1,

Action	Events	Call info
B1 Answers	GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1

Called Party Redirects a Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 redirects the call in connected state.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 redirects the call to C Connection for C is created C rings B1 gets dropped A gets CallStateChg for Ringback	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1 CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = null
C Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = B1, CiscoCall.getCurrentCallingTerminal() = terminal of A. CiscoCall.getCurrentCalledTerminal() = terminal of C

Called Party Which Has Transformed Calling and Called Parties Parks the Call and Receives a Park Reminder Call

Configuration

Phone A, B are in cluster devices. C is a park DN (also in cluster)

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 answers and then B1 parks the call and after park reversion timer expiry receives the reminder call.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 Parks the call Connection for C is created B1 gets park reminder B1 rings C gets dropped B1 Answers	… See use case 15.7.1.1.1 … GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnQueuedEv-C GC1-ConnCreatedEvent-B1 GC1-ConnInprogressEvent-B1 GC1-CallCtlConnOfferedEv-B1 GC1-ConnAlertingEvent-B1 GC1-CallCtlConnAlertingEv-B1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B1

Calling Party Parks the Call and Receives a Park Reminder Call After a Transformation From Called Party Translation Pattern

Configuration

Phone A, B are in cluster devices. C is a park DN

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 answers and then A parks the call and after park reversion timer expiry receives the reminder call.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) A Parks the call Connection for C is created A gets park reminder A rings	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnQueuedEv-C GC1-ConnCreatedEvent-A GC1-ConnInprogressEvent-A GC1-CallCtlConnOfferedEv-A GC1-ConnAlertingEvent-A GC1-CallCtlConnAlertingEv-A GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = A, .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = A, .getCalledAddress() = A, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of B1 .getCurrentCalledTerminal() = null
C gets dropped A Answers	GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = A, .getCalledAddress() = A, .getLastRedirectedAddress() = Park DN C, .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of A

Caller Redirects a Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. A redirects the call in connected state.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) A redirects the call to C Connection for C is created C rings A gets dropped B1 gets CallStateChg for Ringback C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CiscoCall.getModifiedCallingAddress() = A1, CiscoCall.getCallingAddress() = A, CiscoCall.getModifiedCalledAddress() = B1, CiscoCall.getCalledAddress() = B1, CiscoCall.getCurrentCallingTerminal() = Terminal of A. CiscoCall.getCurrentCalledTerminal() = Terminal of B1 CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1. CiscoCall.getCurrentCalledTerminal() = null CiscoCall.getModifiedCallingAddress() = B1, CiscoCall.getCallingAddress() = B1, CiscoCall.getModifiedCalledAddress() = C, CiscoCall.getCalledAddress() = C, CiscoCall.getLastRedirectedAddress() = A, CiscoCall.getCurrentCallingTerminal() = terminal of B1.

Called Party Transfers the Call Which Has Transformed Calling and Called Parties

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 consult transfer the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null

Actions	Events	Call info
Transfer starts Call Changes	GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv	.getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B1, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1

Actions	Events	Call info
Connection for C is created (GC1) C gets dropped (GC2) B1 gets dropped (GC1) B1 gets dropped (GC2) GC2 Invalid Transfer comple	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

Called Party Transfers the Call Which Has Transformed Calling and Called Parties to a DN Which Matches the Translation Pattern with Calling Party Transformation Defined

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

C matches the the translation pattern with calling party transformation to B2

Procedure:

Application invokes connect() at A to call B. B1 consult transfer the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C,

Actions	Events	Call info
Transfer starts Call Changes	GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv	.getCalledAddress() = C, .getLastRedirectedAddress() = null .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B2, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1 .getModifiedCallingAddress() = A1,

Actions	Events	Call info
Connection for C is created (GC1) C gets dropped (GC2) B1 gets dropped (GC1) B1 gets dropped(GC2) GC2 Invalid Transfer complete	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B1 GC1-CallCtlConnDisconnectedEv-B1 GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	.getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

Called Party with Transformed Calling and Called Parties Conferences a DN

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

Procedure:

Application invokes connect() at A to call B. B1 consult conference the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv CG1-CallCtlTermConnHeldEv GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B1, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C

Actions	Events	Call info
Conference Starts B1 gets dropped (GC2)	GC1-CiscoConferenceStartEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC1-CiscoTermConnSelectChangedEv GC1-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv	Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B1, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1
Connection for C is created (GC1) C gets dropped (GC2) GC2 invalid Conferece Ends	GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C * GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv-C GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoConferenceEndEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = null, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Called Party with Transformed Calling and Called Parties Conferences a DN Which Matches the Translation Pattern with Calling Party Transformation Defined

Configuration

Phone A, B, C are in cluster devices.

B has a translation pattern configured where both calling and called parties get transformed to A1 and B1 respectively

C has a translation pattern configured where calling party gets transformed to B2.

Procedure:

Application invokes connect() at A to call B. B1 consult conference the call to C.

Actions	Events	Call info
A and B1 receive Connected Call State (Basic Call) B1 consults call to C Connection for C is created (GC2) C rings C Answers	GC1-ConnConnectedEvent-B1 GC1-CallCtlConnEstablishedEv-B1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CiscoTermConnSelectChangedEv CG1-CallCtlTermConnHeldEv GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEvent-B1 GC2-ConnConnectedEvent-B1 GC2-CallCtlConnInitiatedEv-B1 GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-B1 GC2-CallCtlConnEstablishedEv-B1 GC2-ConnCreatedEvent-C GC2-ConnInprogressEvent-C GC2-CallCtlConnOfferedEv-C GC2-ConnAlertingEvent-C GC2-CallCtlConnAlertingEv-C GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv	.getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = Terminal of B1 .getModifiedCallingAddress() = B1, getCallingAddress() = B1, .getModifiedCalledAddress() = null, .getCalledAddress() = null, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = null

Actions	Events	Call info
Conference Starts B1 gets dropped (GC2)	GC2-ConnConnectedEvent-C GC2-CallCtlConnEstablishedEv-C GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC1-CiscoConferenceStartEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-B1 GC2-CallCtlConnDisconnectedEv-B1 GC1-CiscoTermConnSelectChangedEv GC1-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv	.getModifiedCallingAddress() = B2, .getCallingAddress() = B1, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of B1. .getCurrentCalledTerminal() = terminal of C Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = B2, OCall.getCallingAddress() = B1, OCall.getModifiedCalledAddress() = C, OCall.getCalledAddress() = C, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of B1 OCall.getCurrentCalledTerminal() = terminal of C
Connection for C is created (GC1) C gets dropped (GC2) GC2 invalid Conferece Ends	GC1-ConnCreatedEvent-C * GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv-C GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoConferenceEndEv	FCall.getModifiedCallingAddress() = A1, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = B1, FCall.getCalledAddress() = B1, FCall.getLastRedirectedAddress() = FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of B1 .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = null, .getCalledAddress() = C, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

WildCard Routepoint Interaction (Behavior Change)

WildCard RoutePoint Redirects a Basic Incoming Call to IPPhone

Configuration

Phone A, B are in cluster devices. 4XXX is a wildcard routepoint

Service parameter “Use WildCard pattern in CTI Call Info” is set to true.

Procedure:

Application invokes connect() at A to call 4000. 4XXX redirects the call to B.

Actions	Events	Call info
A initiates call to 4000 Connection of A created, called party info set Connection of 4XXX created 4XXX Redirects to B Connection for B created B is ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-4XXX GC1-ConnInprogressEvent-4XXX GC1-CallCtlConnOfferedEv-4XXX GC1-ConnAlertingEvent-4XXX GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCalledAddress() = 4XXX, .getCurrentCalledAddress() = 4XXX .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4XXX,

Actions	Events	Call info
4XXX gets dropped B Answers	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4XXX GC1-CallCtlConnDisconnectedEv-4XXX GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getLastRedirectedAddress() = 4XXX, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCalledAddress() = 4XXX, .getLastRedirectedAddress() = 4XXX, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B

WildCard Routepoint Interaction (Original Behavior)

WildCard RoutePoint Redirects a Basic Incoming Call to IPPhone

Notes / Caveats

This configuration is not supported. This use case is only intended to show the call flow or events for the above use case with the Use WildCard pattern in CTI Call Info service parameter turned off. Applications should not count on this information to be correct, and to properly support Wildcard Routepoint scenarios, should look to adapting their applications so that they can support the new service parameter being enabled.

An important thing to note is that a connection is created for the dialed DN, 4000. This connection, as well as the connection of 4XXX is not dropped from the call until the redirect happens. This means that if a Wildcard DN is configured on a phone or device, you will see connections for the calling party, 4000, and 4XXX. This basic call will have three connections, which may confuse applications, which might believe it to be a conference call. CiscoCall.isConference() would still return false in this scenario. As stated in previous sections, this extra connection is created in error, and applications should not rely on this connection being there.

Configuration

Phone A, B are in cluster devices. 4XXX is a wildcard routepoint

Service parameter “Use WildCard pattern in CTI Call Info” is set to false / OFF.

Procedure:

Application invokes connect() at A to call 4000. 4XXX redirects the call to B.

Actions	Events	Call info
A initiates call to 4000 Connection of A created, called party info set Connection of 4000 created Connection of 4XXX created	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-4000 GC1-ConnInprogressEvent-4000 GC1-CallCtlConnOfferedEv-4000 GC1-ConnAlertingEvent-4000 GC1-ConnCreatedEvent-4XXX GC1-ConnInprogressEvent-4XXX GC1-CallCtlConnOfferedEv-4XXX	.getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCurrentCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCurrentCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCurrentCalledAddress() = 4000, .getCalledAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
4XXX Redirects to B Connection for B created B is ringing B Answers	GC1-ConnAlertingEvent-4XXX (note: 3 connections on the 2 party call.) GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4XXX GC1-CallCtlConnDisconnectedEv-4XXX GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-4000 GC1-CallCtlConnDisconnectedEv-4000 GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-B GC1-CallCtlConnEstablishedEv-B GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCurrentCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = 4000, .getCurrentCalledAddress() = 4000, .getCalledAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4000, .getLastRedirectedAddress() = 4000, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = 4000, .getLastRedirectedAddress() = 4000, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of B

External Call Control Use Cases

External Call Control on Translation Pattern and CEPM Returns “continue”

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns continue and hence call will be presented to B1 (see use case “Basic Call initiated from JTAPI to the DN with Translation Pattern configured to transform called party” in the Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 171 topic).

External Call Control on Translation Pattern and CEPM Returns “divert”

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divert to C.

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCurrentCalledAddress() = BXXX, .getCalledAddress() = BXXX, .getLastRedirectedAddress() = BXXX .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCurrentCalledAddress() = C,

Actions	Events	Call info
C Answers	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getCalledAddress() = BXXX, .getLastRedirectedAddress() = BXXX .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = BXXX, .getLastRedirectedAddress() = B1 .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns <reject>

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B. The routing rule for B says “Reject”<reject>

CEPM returns reject.

A receives ConnFailedEvent (cause = CtiCallRejected), ConnDisconnectedEv (cause = normal), CallInvalidEvent (caue = Normal).

Actions	Events	Call info
A initiates call to B Connection of A created, CEPM Returns Reject	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnFailedEv-A GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-CallInvalidEvent GC1-CallObservationEndedEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null Cause = CtiCallRejected Cause = Normal

External Call Control on Translation Pattern and CEPM Returns “continue” with Modified Calling and Called Parties

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns continue with ModifiedCalling = “MA” and ModifiedCalled = “MB”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 171.)

Call Events:

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns continue with modified calling/called Connection of MBcreated MB starts ringing BAnswers	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-MB GC1-ConnInprogressEvent-MB GC1-CallCtlConnOfferedEv-MB GC1-ConnAlertingEvent-MB GC1-CallCtlConnAlertingEv-MB GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-MB GC1-ConnConnectedEvent-MB GC1-CallCtlConnEstablishedEv-MB GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getCurrentCallingAddress() = A .getModifiedCalledAddress() = MB, .getCurrentCalledAddess() = MB, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = MB, .getCalledAddress() = MB, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of MB

External Call Control on Translation Pattern and CEPM Returns “divert” with Modified Calling and Called Parties

Configuration:

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result:

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 171.)

Call Events:

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing C Answers	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = MB, .getCurrentCalledAddress() = C .getCalledAddress() = B1, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns “divert” with Modified Calling and Called Parties with resetCallHistory flag = resetLastHop

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”, resetCallHistory = “resetLastHop”

Call will be extended to “C” (based on description for modified calling and modified called in divertTo routing directive, overrides the calling & called number transformation configured for translation pattern and the call is diverted to C. For details, see Use Cases for Calls Going Through Translation Pattern with CEPN Info in Cc Signals, on page 171.)

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
C Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C

External Call Control on Translation Pattern and CEPM Returns "divert" with Modified Calling and Called Parties with resetCallHistory flag = resetAll

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B.

CEPM returns divertTo = "C", with ModifiedCalling = "MA" and ModifiedCalled = "MB"

C has a userRule configured to DivertTo = "D" with ModifiedCalling = "MMA", ModifiedCalled = "MMB", resetCallHistory = "resetAll"

Call will be extended to "D"

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling CEPM Returns divert to D, modify Called/Calling Connection of D created D starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-D GC1-ConnInprogressEvent-D GC1-CallCtlConnOfferedEv-D GC1-ConnAlertingEvent-D GC1-CallCtlConnAlertingEv-D GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-D	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A1, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MMA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCurrentCalledAddress() = D, .getCalledAddress() = B1, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null
D Answers	GC1-ConnConnectedEvent-D GC1-CallCtlConnEstablishedEv-D GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MMA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCalledAddress() = D, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of D

External Call Control on Translation Pattern and CEPM Returns <reject> and Service Parameter CTI Use Wildcard Pattern as calledPartyDN Is Set to False

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX which has calling and called party transformation defined to transform A to A1 and B to B1 and External Call Control is also enabled.

Procedure:

Application invokes connect() at A to call B.

Result

Dialed number B matches the translation pattern BXXX which has External Call Control enabled. This takes precedence and CUCM requests CEPM to get routing rule for B. The routing rule for B says “Reject”<reject>

CEPM returns reject.

Jtapi throws platform exception to the application. A receives ConnFailedEvent (cause = CtiCallRejected), ConnDisconnectedEv (cause = normal), CallInvalidEvent (caue = Normal).

Actions	Events	Call info
A initiates call to B Connection of A created, CEPM Returns Reject	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnFailedEv-A Jtapi throws Exception: PlatformException GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-CallInvalidEvent GC1-CallObservationEndedEv	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null Exception info: Could not meet post conditions of connect() Cause = CtiCallRejected Cause = Normal

Transfer and External Call Control with Modified Calling and Called Parties

Configuration

Phone A, B are in cluster devices. B matches the translation pattern BXXX where External Call Contol is enabled.

Phone C and D does not match any translation pattern, and have no External Call Control defined.

Procedure:

Application invokes connect() at A to call B. CEPM returns divertTo = C, with ModifiedCalling = “MA” and ModifiedCalled = “MB”.

C initiate transfer to D and completes the transfer.

Result

Transfer is successfully completed

Actions	Events	Call info
A initiates call to B Connection of A created, called party info set CEPM Returns divert to C, modify Called/Calling Connection of C created C starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B1, .getCalledAddress() = B1, .getLastRedirectedAddress() = null, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = B1, .getLastRedirectedAddress() = MB .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null

Actions	Events	Call info
C Answers C consult transfer to D Connection for C created (GC2) Connection for D created (GC2) D Answers	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv CG1-CallCtlTermConnHeldEv GC2-ConsultCallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-CallCtlConnEstablishedEv-C GC2-ConnCreatedEvent-D GC2-ConnInprogressEvent-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEvent-D GC2-CallCtlConnAlertingEv-D GC2-TermConnCreatedEvent GC2-TermConnRingingEvent GC2-CallCtlTermConnRingingEv GC2-ConnConnectedEvent-D GC2-CallCtlConnEstablishedEv-D GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	.getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = C, .getCalledAddress() = C, .getLastRedirectedAddress() = MB, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = D, .getCalledAddress() = D, .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C, .getCallingAddress() = C, .getModifiedCalledAddress() = D, .getCalledAddress() = D,

Actions	Events	Call info
Transfer Starts D gets added to GC1	GC1-CiscoTermConnSelectChangedEv GC2-CiscoTermConnSelectChangedEv GC1-CiscoTransferStartEv GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-D GC1-ConnConnectedEvent-D GC1-CallCtlConnEstablishedEv-D GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	.getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C. .getCurrentCalledTerminal() = terminal of D Ev.getOriginalCall = GC2 (OCall) Ev.getSurvivingCall = GC1 (FCall) OCall.getModifiedCallingAddress() = C, OCall.getCallingAddress() = C, OCall.getModifiedCalledAddress() = D, OCall.getCalledAddress() = D, OCall.getLastRedirectedAddress() = OCall.getCurrentCallingTerminal() = terminal of C OCall.getCurrentCalledTerminal() = terminal of D FCall.getModifiedCallingAddress() = MA, FCall.getCallingAddress() = A, FCall.getModifiedCalledAddress() = C, FCall.getCalledAddress() = C, FCall.getLastRedirectedAddress() = MB FCall.getCurrentCallingTerminal() = terminal of A FCall.getCurrentCalledTerminal() = terminal of C .getModifiedCallingAddress() = MA, .getCallingAddress() = A, .getModifiedCalledAddress() = D, .getCurrentCalledAddress() = D, .getCalledAddress() = B1, .getLastRedirectedAddress() = C .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of D

Actions	Events	Call info
D gets dropped from GC2 C gets dropped from GC1 C gets dropped fromGC2 GC2 Invalid Transfer ends	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-D GC2-CallCtlConnDisconnectedEv-D GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-C GC1-CallCtlConnDisconnectedEv-C GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CiscoTransferEndEv	

Chaperone Use Cases

Call Is Redirected to a Hunt List of Chaperones and the Chaperone Enables Call Recording and Conferences in the Called Party

Configuration

A calls X, X's DN matches the translation pattern where External Call Control is enabled.

CEPM determines this call needs to have a chaperone's supervise. CEPM returns the permit decision with the obligation <divert>, destination HuntPilot B, which is a hunt pilot of chaperones, and a reason string "chaperone".

CUCM redirects the call to the hunt pilot B, and the chaperone C1 answers the call.

After talking to A briefly and discovered that A intended to talk to D, the chaperone C1 starts to establish a conference to D. C1 presses the conference softkey and dials D.

CUCM queries CEPM for the call, with calling user C1 with DN C1, and called user D with DN D.

CEPM returns the response with permit decision with <continue> call routing directive, since the policy server detects that the caller is the chaperone.

CUCM rings D's phone and D answers the call.

C1 presses the conference softkey again, and the conference is established.

The chaperone C1 presses the “record” softkey. This triggers the call recording being setup from C1’s IP phone to the recorder.

As one of the steps to establish recording calls to the recorder, two recording calls setup are first sent to the BIB of C1’s IP phone (INVITE for SIP phone and SCCP only the media message are involved). Note only one recording is shown in the picture.

As another step to establish the recording calls to the recorder, the two calls are then redirected to the recorder.

When the call recording is eablished successfully, the recording warning tone is playing to the C1’s phone. The recording warning tone is enabled by setting service parameter “Play Recording Notification Tone To Observed Target” to True.

After comfirming the call recording is established successfully, the chaperone reads an announcement to both A and D and informs them the call is being recorded.

A and D starts to talk under the supervision of the chaperone.

NOTE

Chaperones have limited abilities in what they can do on a call. The most obvious example is that they cannot put the call on hold, because they are required to be on the call at all times. To learn more about Chaperone limitations, please see the related sections of the External Call Control FFS.

Call Events:

Actions	Events	Call info
A initiates call to X Connection of A created, Hunt connection created (see Hunt List section) Connection of C1 created C1 starts ringing	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-CiscoHuntConnCreatedEv-B GC1-ConnInProgressEv-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEv-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C1 GC1-ConnInprogressEvent-C1 GC1-CallCtlConnOfferedEv-C1 GC1-ConnAlertingEvent-C1 GC1-CallCtlConnAlertingEv-C1	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedParty() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X,

Actions	Events	Call info
C1 Answers C1 initiates conference Conference consult call to D , D answers	GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C1 GC1-ConnConnectedEvent-C1 GC1-CallCtlConnEstablishedEv-C1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlTermConnHeldEv-TermC1 GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEv-C1 GC2-CallCtlTermConnTalkingEv-TermC1 GC2-CallCtlConnDialingEv-C1 GC2-CallCtlConnEstablishedEv-C1 GC2-CiscoConnCreatedEv-D GC2-ConnInProgressEv-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEv-D	.getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C1 Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = .getCurrentCalledAddress() = .getCalledAddress() = .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = D .getCurrentCalledAddress() = D .getCalledAddress() = D

Actions	Events	Call info
Call 2 merges Conn for D created on GC1 Call 2 cleaned up Chaperone C1 starts recording Chaperone C1 tries to redirect the call	GC2-CallCtlConnAlertingEv-D GC2-ConnConnectedEv-D GC2-CallCtlConnEstablishedEv D CiscoCallChangedEv final call = GC1, consult call = GC2 GC1-ConnCreatedEv D GC1-ConnConnectedEv D GC1-CallCtlConnEstablishedEvD GC2-ConnDisconntedEv C1 GC2-ConnDisconntedEv D GC2-CallCtlConnDisconnectedEv C1 GC2-CallCtlConnDisconnectedEv D GC2-CallCtlTermConnDroppedEv C1 GC2-CallCtlTermConnDroppedEv D GC2-CallInvalidEv Normal recording events when recording is initiated on a conference call will be received. InvalidStateException : Did not meet pre conditions.	.getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = terminal of D

Call Is Redirected to a Hunt List of Chaperones and the Chaperone Conferences in the Called Party From Application

Configuration

A calls X, X's DN matches the translation pattern where External Call Control is enabled.

CUCM redirect the call to the hunt pilot B. Call is intercepted by the chaperone and the chaperone C1 answers the call.

After talking to A briefly and discovered that A intended to talk to D, the chaperone C1 starts to establish a conference to D. C1 initiates a consult call to D. A new global call id GC2 is created.

CUCM rings D's phone and D answers the call.

C1 invokes GC2.conference(GC1) from application.

At this step, request for establishing the conference would fail. Jtapi would throw InvalidStateException with the error code as "Call state not valid".

In order to establish a conference successfully, application must invoke the conference by passing the CI of the call in which chaperone is the controller as the primary CI. So in this case, if application invokes GC1.conference(GC2), it would be able to establish the conference successfully and if application invokes GC2.conference(GC1), Jtapi would throw an exception.

Also application can use CiscoConnection.isChaperone() API to determine controller is chaperone on which call.

Call Events:

Actions	Events	Call info
A initiates call to X Connection of A created, Hunt connection created (see Hunt List section) Connection of C1 created	GC1-CallActiveEvent GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv-A GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-CiscoHuntConnCreatedEv-B GC1-ConnInProgressEv-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEv-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C1 GC1-ConnInprogressEvent-C1 GC1-CallCtlConnOfferedEv-C1	.getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = “”, .getCalledAddress() = “”, .getLastRedirectedParty() = “”, .getCurrentCallingTerminal() = Terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X,

Actions	Events	Call info
C1 starts ringing C1 Answers C1 initiates conference	GC1-ConnAlertingEvent-C1 GC1-CallCtlConnAlertingEv-C1 GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv-C1 GC1-ConnConnectedEvent-C1 GC1-CallCtlConnEstablishedEv-C1 GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlTermConnHeldEv-TermC1 GC2-CiscoConsultCallActiveEv GC2-ConnCreatedEv-C1 GC2-CallCtlTermConnTalkingEv-TermC1 GC2-CallCtlConnDialingEv-C1 GC2-CallCtlConnEstablishedEv-C1	.getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = null Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = A, .getCallingAddress() = A, .getModifiedCalledAddress() = B, .getCurrentCalledAddress() = B .getCalledAddress() = X, .getLastRedirectedAddress() = X, .getCurrentCallingTerminal() = terminal of A. .getCurrentCalledTerminal() = terminal of C1 Reason = REASON_EXTERNALCALLCONTROL .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = .getCurrentCalledAddress() = .getCalledAddress() = .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1

Actions	Events	Call info
Conference consult call to D , D answers C1 completes the conference by invoking GC2.conference(GC1) from application. C1 tries to complete the conference by invoking GC1.conference(GC2_from application	GC2-CiscoConnCreatedEv-D GC2-ConnInProgressEv-D GC2-CallCtlConnOfferedEv-D GC2-ConnAlertingEv-D GC2-CallCtlConnAlertingEv-D GC2-ConnConnectedEv-D GC2-CallCtlConnEstablishedEv D InvalidStateException : Call state not valid. CiscoCallChangedEv final call = GC1, consult call = GC2 GC1-ConnCreatedEv D GC1-ConnConnectedEv D GC1-CallCtlConnEstablishedEv D GC2-ConnDisconntedEv C1 GC2-ConnDisconntedEv D GC2-CallCtlConnDisconnectedEv C1 GC2-CallCtlConnDisconnectedEv D GC2-CallCtlTermConnDroppedEv C1 GC2-CallCtlTermConnDroppedEv D GC2-CallInvalidEv	.getCurrentCalledTerminal() = null .getModifiedCallingAddress() = C1, .getCallingAddress() = C1, .getModifiedCalledAddress() = D .getCurrentCalledAddress() = D .getCalledAddress() = D .getLastRedirectedAddress() = .getCurrentCallingTerminal() = terminal of C1 .getCurrentCalledTerminal() = terminal of

Extension Mobility Cross Cluster

Actions	Events	Call info
1. User1 has a device profile configured with DN A in cluster1. This profile is included in the control list of application. User1 goes to a visiting cluster and EM login to a device TERMA. Device registers to cluster1	Events to provider observer CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA	CiscoTerminal.getLoginType() returns CiscoTerminal.NO_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN CiscoTerminal.getLoginType() returns CiscoTerminal.VISITOR_LOGIN

Actions	Events	Call info
User1 logs off from the Device	CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT CiscoTerminal.getLoginType() returns CiscoTerminal.NO_LOGIN
2. User1 has a device profile configured with DN A in cluster1. This device profile is included in the control list of application. User1 EM into a device TERMA on cluster1. Device re-registers with DN A. The device TERMA is not in application control listapplication control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN CiscoTerminal.getLoginType() returns CiscoTerminal.NATIVE_LOGIN
User1 log off from the device. Device re-registers to cluster1 with default DN.	CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
3. EM into a controlled Device: User1 has a device profile configured with DN A in cluster1. This device profile is included in the control list of application. User1 EM into a device TERMA on cluster1. TERMA with default DN X is in application control list. Device re-registers with DN A.	CiscoAddrRemovedEv X CiscoAddrCreatedEv A	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN
User1 logs out of the device. Device Unregister, Device and line out of service Device Register to CM with default DN X.	CiscoAddrRemovedEv A CiscoAddrCreatedEv X	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
4. Application uses user1 userid. Device profile agent1 is added to application control list after application is started User EMs into a device TERMA and gets the device profile. Agent1 device profile is removed from application control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_REMOVE getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE

Actions	Events	Call info
5. EMCC scenario resulting in CiscoAddrAddedToTerminalEv and CiscoAddrRemovedFromTerminalEv Cluster1 has application with Terminal TermA (address A) in control list. User1 is a device profile which is configured with line A is included in app control list. User goes to a visiting cluster and logs into a device (TermX, Addr X). TermX registers with cluster1 with address A User logs out of device TermX	CiscoTermCreatedEv TERMX CiscoAddrAddedToTerminalEv AddrA CiscoAddrRemovedFromTerminalEv AddA CiscoTermRemovedEv TERMx	getCiscoCause() returns CiscoProvEv.CAUSE_EM getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
6. Device profile Agent1 with DN A is logged into a device TERMA. User1 opens provider and then adds the profile Agent1 to the control list through the admin pages. User1 then removes the profile from the control list	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_ADD getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGIN_ PROFILE_ADD getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE getCiscoCause() returns CiscoProvEv. CAUSE_EM_LOGOUT_ PROFILE_REMOVE
7. Device profile Agent1 is not in the applications control list but it is there as a controlled profiles for extension mobility for user1. User1 opens provider and logs into terminal TERMA with profile agent1 and the same user id with which it had opened the provider. User1 logs out of the device	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT
8. Device profile Agent1 (DN A) is in the applications control list with user as user1. User1 opens the provider and does an EM login into TERMA with profile as agent1. TERMA is not in control list. User1 logs out of TERMA.	CiscoAddrCreatedEv A CiscoTermCreatedEv TERMA CiscoAddrRemovedEv A CiscoTermRemovedEv TERMA	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT

Actions	Events	Call info
9. Device profile Agent1 (DN A) is in the applications control list with user as user1. User1 opens the provider and does an EM login into TERMA with profile as agent1. TERMA is in control list with default DN as X. User1 logs out of TERMA.	CiscoAddrRemovedEv X CiscoAddrCreatedEv A CiscoAddrRemovedEv A CiscoAddrCreatedEv X	getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGIN getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT getCiscoCause() returns CiscoProvEv.CAUSE_EM_LOGOUT

End to End Session ID for Calls

Session ID in a Basic Call Scenario

Application has already opened provider and observing TermA and TermB

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA	
	GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A	((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnA) = A's LocalUUID

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB	
	GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnA) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnB) = A's LocalUUID
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev.getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev.getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

SessionID for a Basic Call involving SIP Endpoints

Application has already opened provider and observing SIP terminals TermA and TermB

Action	Events	Call information
Application makes a call between TermA and TermB	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = null ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = null ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = A's local UUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = B's localUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = null ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID

Action	Events	Call information
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

SessionIDs for Calls Involving Shared Lines and Hold Resume

B and B are lines shared on two terminals. Application has opened provider and observed A, B and B

Action	Events	Call information
Application makes a call between TermA and TermB		((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnA) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnA) = B's or (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID

Action	Events	Call information
	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv	

Action	Events	Call information
	TermB GC1 TermConnCreatedEv TermB' GC1 TermConnRingingEv TermB' GC1 CallCtlTermConnRingingEv TermB'	
B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv TermB GC1 TermConnPassiveEv TermB' GC1 CallCtlTermConnBridgedEv TermB'	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

Action	Events	Call information
B puts the call on hold	GC1 TermConNActiveEv TermB GC1 CallCtlTermConnHeldState TermB GC1 TermConnActiveEv termB' GC1 CallCtlTermConnHeldState TermB'	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB') = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB') = A's localUUID

Action	Events	Call information
B' resumes the call	GC1 TermConnActiveEv TermB' GC1 CallCtlTermConnTalkingEv TermB' GC1 TermConnPassiveEv TermB GC1 CallCtlTermConnBridgedEv TermB	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID ((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB') = (B')'s LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB') = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB') = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = (B')'s localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB') = A's localUUID

SessionID when Call is Redirected to a Third Party

Application has already opened provider and observed A,B and C and establishes a call between A and B.

Action	Events	Call information
A calls B and B answers	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConNActiveEv B GC1 CallCtlTermConnTalkingEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnB) = B's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnB) = A's LocalUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getLocalUUID(termConnB) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = B's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(B)).getPeerUUID(termConnB) = A's localUUID

Action	Events	Call information
Application redirects the call from B to C	GC1 CallActiveEv C GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C GC1 TermConnCreatedEv TermC GC1 TermConnActiveEV TermC GC1 CallCtlTermConnTalkingEv TermC GC1 CallCtlConnDialingEv C GC1 CallCtlConnEstablishedEv C GC1 TerConnDroppedEv TermB CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	((CiscoConnection)Ev,getConnection()).getLocalUUID(termConnC) = C's LocalUUID ((CiscoConnection)Ev,getConnection()).getPeerUUID(termConnC) = A's LocalUUID CallInfo : CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getLocalUUID(termConnA) = A's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(C)).getLocalUUID(termConnC) = C's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(A)).getPeerUUID(termConnA) = C's localUUID (CiscoConnection)(CiscoProvider.getCall(gcid, cid).getConnection(C)).getPeerUUID(termConnC) = A's localUUID

Forced Authorization and Customer Matter Codes

Scenario One

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

Action	Event
A calls B by using call.Connect(), or A places a consult call to B by using Call.Consult().	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv. FAC_REQUIRED
Application enters additional digits by using CiscoConnection.addToAddress.	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv BTermConnRingingEv B CallCtlTermConnRingingEv B ConnConnectedEv B CallCtlConnEstablishedEv B
B answers the call.	TermConnActiveEv B

Scenario Two

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

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

Action	Event
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B

Scenario Three

The application controls A and B;

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

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

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

Scenario Four

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

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

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

Scenario Five

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

A and B are PNO and within the Cisco Unified Communications Manager cluster.

Action	Event	Fields
Call arrives at RP	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A)
Application invokes selectRoute(routeselected[], callingsearchspace, modifiyingcallingnumber[], preferredOriginalCdNumber[], preferredOriginalCdOption[], facCode[], cmcCode[]) where routeSelected[] = BcallingSearchSpace = CiscoRouteSession.DEFAULT_SEARCH_SPACEmodifyingCgNumber = null, preferredOriginalCdNumber = null, preferredOriginalCdOption = CiscoRouteSession.DONOT_RESET_ORIGINALCALLED, facCode[] = “facCode for B”cmcCode[] = “cmcCode for B”	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA (Terminal associated with A) getRouteUsed () = B
Application invokes endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_ENDgetRouteAddress () = RP

Hairpin Support

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

S.No.	Pre-Condition	Use Case	Expected Behavior	Result
3	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A and B.	A calls B. B does a conference call to C via gateway. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
4	IP Phones A and B are in same cluster, IP phone C is in another cluster. JTAPI observes A, B and C.	A calls B. B does a conference call to C via gateway. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
5	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway is able to pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A, B and C are in conference call.
6	IP Phones A, B, C are in same cluster, IP phone D is in another cluster. JTAPI observes A, B and C. Gateway does not pass new party information to each other.	A calls B. B does a conference call to D via gateway. D transfers the call to C. B completes the conference and all A, B and C are in conference.	At A and B, ConferenceCallStateChanged event has participantInfo with following types: A: CiscoAddress.Internal B: CiscoAddress.Internal D: CiscoAddress.External	A, B and C are in conference call.
7	IP Phones A and C are in same cluster, IP phone B is in another cluster. JTAPI observes A and C.	A calls B via gateway. B redirects call to C. Now IP Phones A and C are connected.	At A: It is connected to C. A's type is CiscoAddress.Internal C's type is CiscoAddress.External At C: It is connected to A. C's type is CiscoAddress.Internal A's type is CiscoAddress.External	A and C are connected.

Half Duplex Media

RTP Event at A and B.

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

Hunt List

Configuration

- HuntList feature is enabled for all use cases, unless otherwise indicated
- HuntList pilot1 : 2000
- HuntList1 LineGroup Member : 3001, 3002, 3003
- HuntList pilot2: 4000
- HuntList2 LineGroup Member : 5001, 5002, 5003

Cisco Hunt Address mentioned in the call models below indicates that CiscoAddress.getType() returns CiscoAddress.HUNT_PILOT.

Scenario 1

Scenario	Result
A (1000) calls Hunt Pilot B (2000), Application is observing only A. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA CallingParty = A, current calling = A Called Party = null, current called = null Lrp = null GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000 type = CiscoAddress.HUNT_PILOT

CurrentCalledParty = 2000 type = CiscoAddress.HUNT_PILOT

LastRedirectingParty = Null

Current called display name = 2000Name.

Scenario 2

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001); application is observing A. GC1 is the GCID of the call. C(3001) answers the call	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA CallingParty = A, current calling = A Called Party = null, current called = null Lrp = null GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Current called party display name = 3001Name. Called party display name changes to the display name of the hunt pilot member that answered the call.

Scenario 3

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001). Application is observing C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC:

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 4

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null \

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 5

Scenario	Result
A (1000) calls Hunt Pilot (B or 2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1:CallCtlTermConnTalkingEv TermC

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 6

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001). Application is observing A and C. GC1 is the GCID of the call. A redirects the call to another Hunt Pilot D(4000)	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC CallingParty = A, current calling = A Called Party = B, current called = B Lrp = null GC1:CallCtlTermConnTalkingEv TermC GC1: CiscoHuntConnCreatedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnDroppedEv TA GC1: CallCtlTermConnDroppedEv TA getCallControlCause () = CAUSE_REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause () = CAUSE_REDIRECTED

JTAPI CallInfo

CallingParty = 2000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 7

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) Application is observing A and C. GC1 is the GCID of the call. A redirects the call to D(4000). E(5001) answers the call	….. ….. GC1: CiscoHuntConnCreatedEv D GC1: TermConnDroppedEv TermA GC1: CallCtlTermConnDroppedEv TermA getCallControlCause () = CAUSE_REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause () = CAUSE_REDIRECTED GC1: ConnCreatedEv E GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 8

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001). Application is observing A, E and C. GC1 is the GCID of the call. A redirects the call to D(4000). E(5001) answers the call	….. ….. GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E getCallControlCause () = CAUSE_REDIRECTED GC1: CiscoHuntConnCreatedEv D GC1: TermConnDroppedEv TermA GC1: CallCtlTermConnDroppedEv TermA GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A GC1: CallCtlConnEstablishedEv E

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 2000

CalledParty = 2000

CurrentCalledParty = 4000

LastRedirectingParty = 1000

Scenario 9

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001) and D (3002). Application is observing A, C and D. GC1 is the GCID of the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: CallCtlConnAlertingEv C GC1: CallCtlTermConnCreatedEv TermC GC1: CallCtlTermConnRingingEv TermC GC1: CallCtlConnAlertingEv D GC1: CallCtlTermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 10

Scenario	Result
A (1000) calls Hunt Pilot B(2000), call is offered at C (3001) and D (3002). Application is observing A, C and D. GC1 is the GCID of the call. D answers the call	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: CallCtlConnAlertingEv C GC1: CallCtlTermConnCreatedEv TermC GC1: CallCtlTermConnRingingEv TermC GC1: CallCtlConnAlertingEv D GC1: CallCtlTermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD GC1: CallCtlTermConnTalkingEv TermD GC1: CallCtlTermConnDroppedEv TermC GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 11

Scenario	Result
A (1000) calls Hunt Pilot (B or 2000), call is offered at C (3001) and D (3002). Application is observing A and D. GC1 is the GCID of the call. C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnCreatedEv D GC1: ConnOfferedEv D …. GC1: TermConnCreatedEv TermD GC1: CallCtlTermConnRingingEv TermD GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: CallCtlTermConnDroppedEv TermD GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = 1000

CalledParty = 2000

CurrentCalledParty = 2000

LastRedirectingParty = Null

Scenario 12

Scenario	Result
A (1000) calls Hunt Pilot B (2000), call is offered at C (3001) and is answered. A consults with D (4000) and call is offered at E(5001). A completes the conference. Application is observing A. Initially connection is created to an address with DN = B type = UNKNOWN GC1 is the GCID of the final call. GC2 is the consult call C answers the call E answers the call Conference is completed	

Scenario	Result
	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B-U GC1: ConnInProgressEv B-U GC1: CallCtlConnOfferedEv B-U GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnDisconnectedEv B-U GC1: CallCtlConnDisconnectedEv B-U GC1: ConnCreatedEv C GC1: ConnOfferedEv C …. GC1: CallCtlConnEstablishedEv C GC1: CallCtlTermConnHeldEv TermA GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv A ….. GC2: CallCtlTermConnTalkingEv TermA GC2: CallCtlConnDialingEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoHuntConnCreatedEv D GC2: ConnInProgressEv D GC2: CallCtlConnOfferedEv D GC2: ConnAlertingEv D

Scenario	Result
	GC2: CallCtlConnAlertingEv D GC2: ConnCreatedEv E GC2: ConnConnectedEv E GC2: CallCtlConnEstablishedEv E GC2: ConnConnectedEv D GC2: CallCtlConnEstablishedEv D CiscoCallChangedEv final call –GC1, consult call = GC2 GC1: CiscoHuntConnCreatedEv B GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: CiscoHuntConnCreatedEv E GC1: ConnCreatedEv E GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC2: ConnDisconntedEv B GC2: ConnDisconntedEv C GC2: CallCtlConnDisconnectedEv C GC2: ConnDisconntedEv D GC2: ConnDisconntedEv E GC2: CallCtlConnDisconnectedEv E GC2: CallCtlTermConnDroppedEv TermA .. GC2: ConnDisconntedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv

JTAPI CallInfo

CallingParty = 1000

CurrentCallingParty = [No guaranted for conference scenario]

CalledParty = 2000

CurrentCalledParty = [No guaranted for conference scenario]

LastRedirectingParty = 1000

Scenario 13

Transfer to a line group member.

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A, B and C. GC1 is the GCID of the final call. GC2 is the consult call B answers the call B consults to Hunt pilot C answers the call Transfer is completed	

Scenario	Result
	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv B GC1: ConnOfferedEv B … GC1: TermConnRingingEv TermB GC1: TermConnTalkingEv TermB GC1: CallCtlTermConnHeldEv TermB GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv B ….. GC2: CallCtlTermConnTalkingEv TermB GC2: CiscoHuntConnCreatedEv P GC2: ConnInProgressEv P GC2: CallCtlConnOfferedEv P GC2: ConnCreatedEv C GC2: ConnOfferedEv C GC2: TermConnRingingEv TermC GC2: ConnConnectedEv P GC2: CallCtlConnEstablishedEv P GC2: CiscoHuntConnCreatedEv P GC2: ConnInProgressEv P GC2: CallCtlConnOfferedEv P GC2: ConnAlertingEv P GC2: CallCtlConnAlertingEv P GC2: ConnDisconnectedEv P GC2: CallCtlConnDisconnectedEv P

Scenario	Result
	GC2: ConnConnectedEv P GC2: CallCtlConnEstablishedEv P GC2: ConnConnectedEv C GC2: CallCtlConnEstablishedEv C GC2: TermConnTalkingEv TermC GC1: CiscoHuntConnCreatedEv P GC1: ConnInProgressEv P GC1: CallCtlConnOfferedEv P GC1: ConnAlertingEv P GC1: CallCtlConnAlertingEv P CiscoCallChangedEv final call –GC1, consult call = GC2 GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnTalkingEv TC GC1: ConnConnectedEv P GC1: CallCtlConnEstablishedEv P GC2: ConnDisconntedEv P GC2: CallCtlConnDisconnectedEv P GC2: ConnDisconntedEv B GC2: CallCtlConnDisconnectedEv B … …. GC2: ConnDisconntedEv C GC2: CallCtlConnDisconnectedEv C .. GC2: CallInvalidEv GC1: ConnDisconntedEv B GC1: CallCtlConnDisconnectedEv B GC1: TermConnDroppedEv TB GC1: CallCtlTermConnDroppedEv TB

Scenario 14

Pickup from line group

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. C and D are in the same pickup group. D picks up the call ringing at C. GC2 is the initial call at D. A and D are connected on GC1 D goes off-hook and answers call from C.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC2: CallActiveEv GC2: ConnCreatedEv D GC2: ConnConnectedEv D GC2: CallCtlConnInitiatedEv D GC2: TermConnCreatedEv TD GC2: TermConnActiveEv TD GC2: CallCtlTermConnTalkingEv TD GC2: CiscoCallChangedEv GC2->GC1 GC1: ConnCreatedEv D GC1: ConnConnectedEv D GC1: CallCtlConnInitiatedEv D GC1: TermConnCreatedEv TD GC1: TermConnActiveEv TD GC1: CallCtlTermConnTalkingEv TD GC2: TermConnDroppedEv TD GC2: CallCtlTermConnDroppedEv TD GC2: ConnDisconnectedEv D

Scenario	Result
	GC2: CallCtlConnDisconnectedEv D GC2: CallInvalidEv GC1: TermConnDroppedEv TC GC1: TermConnTermConnDroppedEv TC GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B

Note Note: For this scenario, if the pickup is done from the address of the hunt member that is currently ringing with Auto Pickup disabled, then getCiscoHuntConnection() returns the connection to the hunt pilot. If the pickup is done from an address that is in the pickup group but is not the current ringing terminal, then getCiscoHuntConnection() returns null. If Auto Pickup is enabled, then getCiscoHuntConnection() always returns null after the call is picked up (it does not matter whether the pickup is done from the ringing terminal or from another address in the pickup group). This is true for Pickup, Group Pickup, Other Pickup, and Directed Call Pickup.

Scenario 15

Gpickup a ringing hunt list member.

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. D picks up the call ringing at C. GC2 is the initial call at D. A and D are connected on GC1	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B

Scenario	Result
D goes off-hook and dials the pickup number Z.	

Scenario	Result
	GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC2: CallActiveEv GC2: ConnCreatedEv D GC2: ConnConnectedEv D GC2: CallCtlConnInitiatedEv D GC2: TermConnCreatedEv TD GC2: TermConnActiveEv TD GC2: CallCtlTermConnTalkingEv TD GC2: CallCtlConnDialingEv D GC2: ConnCretatedEv Z GC2: ConnInProgressEv Z GC2: CallCtlConnOfferedEv Z GC2: CallCtlConnEstablishedEv D GC2: CiscoCallChangedEv GC2->GC1 GC1: ConnCreatedEv D GC1: ConnCreatedEv Z GC1: ConnConnectedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnCreatedEv TD GC1: TermConnActiveEv TD GC1: CallCtlTermConnTalkingEv TD GC1: ConnInProgressEv Z GC1: CallCtlConnOfferedEv Z GC2: ConnDisconnectedEv Z GC2: CallCtlConnDisconnectedEv Z GC2: TermConnDroppedEv TD GC2: CallCtlTermConnDroppedEv TD GC2: ConnDisconnectedEv D

Scenario	Result
	GC2: CallCtlConnDisconnectedEv D GC2: CallInvalidEv GC1: ConnDisconnectedEv Z GC1: CallCtlConnDisconnectedEv Z GC1: TermConnDroppedEv TC GC1: TermConnTermConnDroppedEv TC GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B

Note For this scenario, if the pickup is done from the address of the hunt member that is currently ringing with Auto Pickup disabled, getCiscoHuntConnection() returns the connection to the hunt pilot. If the pickup is done from an address that is in the pickup group but is not the current ringing terminal, getCiscoHuntConnection() returns null. If the Auto Pickup is enabled, getCiscoHuntConnection() always returns null after the call is picked up (it does not matter whether the pickup is done from the ringing terminal or from another address in the pickup group). This is true for Pickup, Group Pickup, Other Pickup, and Directed Call Pickup.

Scenario 16

Redirect by a hunt member:

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). Application is observing A, C and D. C redirects the call to D. D is not a member.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnEstablishedEv B GC1: ConnAlertingEv B

Scenario	Result
C answers the call. Application redirects the call from C to D	GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: CallCtlEstablishedEv C GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D getCallControlCause() = CAUSE.REDIRECTED GC1: CallCtlConnDisconnectedEv B GC1: CallCtlConnDisconnected C GC1: TermConnDisconnEv C GC1: CallCtlTermConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called D LRP B type CiscoAdress.UNKNOWN

Scenario 17

Calls Moving Between Members

When call is moving between hunt members, the call could go to invalid state.

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call is offered at D. Application is observing C and D Call moves to D.	GC1: CallActiveEv … GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: CallCtlConnEstablishedEv B GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv TC getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEvTC GC1: CallCtlConnDisconnectedEv TC getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called D LRP = null

Scenario 18

Not All Members Are Observed

If all members are not observed call could go to invalid state when moving between hunt members

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E where it is answered. C, D, E and F are the members. Application is observing C and E.	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC
Call moves to D (not unobserved).	GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv A GC1: CallCtlConnDisconnectedEv A getCallControlCause() = CAUSE.REDIRECTED GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: CallInvalidEv GC1: CallActiveEv

Scenario	Result
Call moves from D to E.	GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE
E answers the call.	GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC1: TermConnActiveEv TE GC1: CallCtlTermConnTalkingEv TE Call info: Current calling A Current Called E LRP = null

Scenario 19

Not All Members Are Observed, but Calling Party Is Observed

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E where it is answered. C, D, E and F are the members. Application is observing A, C and E. Call moves to D (not unobserved). Call moves from D to E. E answers the call.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1; ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE

Scenario	Result
	GC1: CallCtlTermConnRingingEv TE GC1: ConnConnectedEv E GC1: CallCtlConnEstablishedEv E GC1: TermConnActiveEv TE GC1: CallCtlTermConnTalkingEv TE Call info: Current calling A Current Called E LRP = null

Scenario 20

Calling and All Hunt List Members Are Observed; the Call Is Not Answered and Goes to Hunt No Answer Forward Destination

Scenario	Result
A (1000) calls (GC1) Hunt Pilot B (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E. The call goes to hunt no answer forward destination F which is observed. C, D, E and F are the members.Application is observing A, C, D, E and F. Call moves to D. Call moves from D to E. Call moves to F.	

Scenario	Result
	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1; ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C

Scenario	Result
	GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE GC1: TermConnDroppedEv TD GC1: CallCtlTermConnDroppedEv TD getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv F GC1: ConnInProgressEv F GC1: CallCtlConnOfferedEv F GC1: ConnAlertingEv F GC1: CallCtlConnAlertingEv F GC1: TermConnRingingEv TF GC1: CallCtlTermConnRingingEv TF GC1: ConnDisconnectedEv B GC1: CallCtlConnDisconnectedEv B GC1: TermConnDroppedEv TE GC1: CallCtlTermConnDroppedEv TE getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv E GC1: CallCtlConnDisconnectedEv E getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called F

Scenario	Result
	LRP = null

Scenario 21

Forward Hunt No Answer to Another Hunt Pilot

Scenario	Result
A (1000) calls (GC1) Hunt Pilot HP1 (2000), call is offered at C (3001). C does not answer the call, call moves to D, and to E. The call goes to hunt no answer forward destination F which is observed. C, D, E are the members . HP2 is the forward hunt no answer destination. H, L are its members. All parties are observed.	GC1: CallActiveEv GC1: ConnCreatedEv A GC1: ConnConnectedEv A GC1: CallCtlConnInitiatedEv A GC1: TermConnCreatedEv TA GC1: TermConnActiveEv TA GC1: CallCtlTermConnTalkingEv TA GC1: CallCtlConnDialingEv A GC1: CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv HP1 GC1; ConnInProgressEv HP1 GC1: CallCtlConnOfferedEv HP1 GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv HP1 GC1: CallCtlConnEstablishedEv HP1

Scenario	Result
Call moves to D Call moves from D to E.	GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1: TermConnCreatedEv TD GC1: TermConnRingingEv TD GC1: CallCtlTermConnRingingEv TD GC1: TermConnDroppedEv TC GC1: CallCtlTermConnDroppedEv getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C getCallControlCause() = CAUSE.REDIRECTED GC1: ConnCreatedEv E GC1: ConnInProgressEv E GC1: CallCtlConnOfferedEv E GC1: ConnAlertingEv E GC1: CallCtlConnAlertingEv E GC1: TermConnRingingEv TE GC1: CallCtlTermConnRingingEv TE GC1: TermConnDroppedEv TD GC1: CallCtlTermConnDroppedEv TD getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv D GC1: CallCtlConnDisconnectedEv D getCallControlCause() = CAUSE.REDIRECTED

Scenario	Result
Call goes t HP2, call is offered at H Call moves from H to L	GC1: ConnCreatedEv H GC1: ConnInProgressEv H GC1: CallCtlConnOfferedEv H GC1: CiscoHuntConnCreatedEv HP2 GC1: ConnAlertingEv H GC1: CallCtlConnAlertingEv H GC1: TermConnRingingEv TH GC1: CallCtlTermConnRingingEv TH GC1: ConnConnectedEv HP2 GC1: CallCtlConnEstablishedEv HP2 GC1: ConnCreatedEv L GC1: ConnInProgressEv L GC1: CallCtlConnOfferedEv L GC1: ConnAlertingEv L GC1: CallCtlConnAlertingEv L GC1: TermConnRingingEv TL GC1: CallCtlTermConnRingingEv TL GC1: ConnDisconnectedEv HP1 GC1: CallCtlConnDisconnectedEv HP1 GC1: TermConnDroppedEv TH GC1: CallCtlTermConnDroppedEv TH getCallControlCause() = CAUSE.REDIRECTED GC1: ConnDisconnectedEv H GC1: CallCtlConnDisconnectedEv H getCallControlCause() = CAUSE.REDIRECTED Call info: Current calling A Current Called F LRP = null

Scenario 22

Consult Transfer by a Member to Another Hunt Pilot

Scenario	Result
A (1000) calls (GC1) Hunt Pilot HP1 (2000), call is offered at C (3001). C answers the call and consult to HP2. L in HP2 is ringing. C completes the transfer. C and L are observed C answers the call C consults with HP2 (GC2)	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv HP1 GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv HP1 GC1: CallCtlConnEstablishedEv HP1 GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1: TermConnRingingEv TC GC1: CallCtlTermConnRingingEv TC GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnActiveEv TC GC1: CallCtlTermConnTalkingEv TC GC1: CiscoTermConnSelectChangedEv TC GC1: CallCtlTermConnHeldEv TC GC2: ConsultCallActive GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnInitiatedEv C GC2: TermConnCreatedEv C GC2: TermConnActiveEv C GC2: CallCtlTermConnTalkingEv TC GC2: CallCtlConnDialingEv TC GC2: CallCtlConnEstablishedEv TC GC2: CiscoHuntConnCreatedEv HP2 GC2: ConnInProgressEv HP2 GC2: CallCtlConnOfferedEv HP2

Scenario	Result
Call is offered to L C completes the transfer	GC2: ConnCreatedEv L GC2: ConnInProgressEv L GC2: CallCtlConnOfferedEv L GC2: ConnAlertingEv L GC2: CallCtlConnAlertingEv L GC2: TermConnRingingEv TL GC2: CallCtlTermConnRingingEv TL GC2: ConnConnectedEv HP2 GC2: CallCtlConnEstablishedEv HP2 GC2: CiscoCallChangedEv GC1: ConnCreatedEv L GC1: ConnAlertingEv L GC1: CallCtlConnAlertingEv L GC1: TermConnCreatedEv TL GC1: TermConnRingingEv TL GC1: CallCtlTermConnRingingEv TL GC2: ConnDisconnectedEv HP2 GC2: CallCtlConnDisconnectedEv HP2 GC2: TermConnDroppedEv TL GC2: CallCtlTermConnDroppedEv TL GC2: ConnDisconnectedEv L GC2: CallCtlConnDisconnectedEv L GC2: TermConnDroppedEv C GC2: CallCtlTermConnDroppedEv C GC2: ConnDisconnectedEv C GC2: CallCtlConnDisconnectedEv C GC1: ConnDisconnectedEv HP1 GC1: CallCtlConnDisconnectedEv HP1 GC2: CallInvalidEv GC1: CiscoHuntConnCreatedEv HP2 GC2: ConnConnectedEv HP2 GC2: CallCtlConnEstablishedEv HP2

Scenario	Result
L answers the call	GC1: TermConnDroppedEv C GC1: CallCtlTermConnDroppedEv C GC1: ConnDisconnectedEv C GC1: CallCtlConnDisconnectedEv C GC1: ConnConnectedEv L GC1: CallCtlConnEstablishedEv L GC1: TermConnActiveEv L GC1: CallCtlTermConnTalkingEv TL

The following call scenarios are generally un-supported and applications are encouraged to enable the huntlist feature and adapt to the event flows described above.

Following are the expected events when the feature is **disabled**.

Scenario 23

Hunt list feature is disabled.

Basic call to hunt pilot

Scenario	Result
A (1000) calls Hunt Pilot P (2000), call is offered at C (3001) and is answered. Application is observing A. GCID is the call is GC1. C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv P GC1: ConnOfferedEv P … GC1: ConnAlertingEv P GC1: ConnConnected P GC1: CallCtlConnEstablishedEv A

bvHunt list feature is disabled

Scenario 24

Consult – Transfer Scenario

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A and B. GC1 is the GCID of the final call. GC2 is the consult call B answers the call B consults to Hunt pilot C answers the call	GC1:CallActiveEv GC1:ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1: CallCtlConnEstablishedEv A GC1: ConnCreatedEv B GC1: ConnOfferedEv B … GC1: TermConnRingingEv TermB GC1: TermConnTalkingEv TermB GC1: CallCtlTermConnHeldEv TermB GC2: CiscoConsultCallActiveEv GC2: ConnCreatedEv B ….. GC2: CallCtlTermConnTalkingEv TermB GC2: ConnCreatedEv P .. .. GC2: ConnAlertingEv P GC2: CallCtlConnEstablishedEv P

Scenario	Result
Transfer is completed	CiscoTransferStartEv final call –GC1, consult call = GC2 GC1: ConnCreatedEv P CiscoCallChangedEv GC2 = >GC1 GC2: ConnDisconnectedEv P GC2: ConnDisconntedEv B GC2: CallCtlConnDisconnectedEv B … …. GC2: CallInvalidEv GC1: CallCtlConnEstablishedEv P GC1: ConnDisconnectedEv B CiscoTransferEndEv

Scenario 25

Hunt list feature is disabled

Consult – Transfer Scenario

Scenario	Result
A (1000) calls B(1001), consult to Hunt Pilot P (2000), call is offered at C (3001) and is answered. B completes the transfer. Application is observing A, B and C.	UnSupported Configuration

Hunt List Connected Number

Hunt pilot B configured with "Display Line Group Member DN as Connected Party" enabled. B has HL1 as its hunt list which has C and D as its hunt members

Scenario	Expected results	Call info
A calls B, Application is observing A only. GC1 is the GCID of the call. Call is offered on the huntmember C and C answers	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1:CallCtlConnEstablishedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnCreatedEv C GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B	Call.getCurrentCallingAddress = A Call.getModifiedCallingAddress = A Call.getCurrentCalledAddress = null Call.getModifiedCalledAddress = null Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C. Application is observing C. GC1 is the GCID of the call. C answers the call	GC1: CallActiveEv GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: ConnCreatedEv A GC1: CiscoHuntConnCreatedEv B GC1: ConnConnectedEv A GC1: CallCtlConnEstablishedEv A GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC: GC1: ConnConnectedEv C GC1: CallCtlConnEstablishedEv C GC1: TermConnActiveEv TermC CG1: CallCtlTermConnTalkingEv termC	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C Application is observing A and C. GC1 is the GCID of the call. C answers the call.	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1:CallCtlTermConnTalkingEv TermC	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C and then to D Application is observing A, C and D. GC1 is the GCID of the call. Call moves to D and D answers	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1: ConnCreatedEv D GC1: ConnAlertingEv D GC1: ConnDisConnEv C GC1: CallCtlConnDiscConnEv C GC1:TermConnDroppedEv TermC GC1: CallCtlTermConnDroppedEv TermC GC1: ConnConnectedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnActiveEv TermD CG1: CallCtlTermConnTalkingEv termD	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
A calls Hunt Pilot B, call is offered at C Application is observing A, C and D. GC1 is the GCID of the call. C answers the call. C consults D and completes transfer	GC1: CallActiveEv GC1: ConnCreatedEv A ... GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv C GC1: ConnInProgressEv C GC1: CallCtlConnOfferedEv C GC1: CiscoHuntConnCreatedEv B GC1: ConnInProgressEv B GC1: CallCtlConnOfferedEv B GC1: ConnAlertingEv B GC1: CallCtlConnAlertingEv B GC1: ConnAlertingEv C GC1: CallCtlConnAlertingEv C GC1:TermConnCreatedEv TermC GC1: TermConnRingingEv TermC GC1: CallCtlTermConnRingingEvTermC GC1:CallCtlTermConnTalkingEv TermC GC2: CallActiveEv GC2: ConnCreatedEv C ... GC2: CallCtlTermConnTalkingEv A	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Scenario	Expected results	Call info
	GC2: ConnCreatedEv D GC2: ConnInProgressEv D GC2: CallCtlConnOfferedEv D ... GC2: ConnAlertingEv D GC2: CallCtlConnAlertingEv D GC2:TermConnCreatedEv TermD GC2: TermConnRingingEv TermD GC2:CallCtlTermConnRingingEvTermD GC2:CallCtlTermConnTalkingEv TermD CiscoTransferStartEv GC1 ConnCreatedEv D GC1: CallCtlConnEstablishedEv D GC1:CallCtlTermConnTalkingEv TermD ... ... GC1 ConnDroppedEv C GC1 ConnDroppedEv B ... GC2 CallInvalidEv CiscoTransferEndEv	Call.getCurrentCallingAddress = C Call.getModifiedCalledAddress = C Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
A calls D. D calls HP B, call is offered on C	GC1: CallActiveEv GC1: ConnCreatedEv A … GC1: CallCtlTermConnTalkingEv A GC1: ConnCreatedEv D GC1: ConnInProgressEv D GC1: CallCtlConnOfferedEv D GC1: ConnAlertingEv D GC1: CallCtlConnAlertingEv D GC1:TermConnCreatedEv TermD GC1: TermConnRingingEv TermD GC1: CallCtlTermConnRingingEvTermD GC1 CallCtlConnEstablishedEv D GC1:CallCtlTermConnTalkingEv TermD GC2: CallActiveEv GC2: ConnCreatedEv D ... GC2: CallCtlTermConnTalkingEv D GC2: ConnCreatedEv C GC2: ConnInProgressEv C GC2: CallCtlConnOfferedEv C GC2: ConnCreatedEv D GC2: CiscoHuntConnCreatedEv B GC2: ConnConnectedEv D GC2: CallCtlConnEstablishedEv A GC2: ConnConnectedEv B GC2: CallCtlConnEstablishedEv B	Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = D Call.getModifiedCalledAddress = D

Scenario	Expected results	Call info
C answers the call D completes transfer	GC2: CallCtlConnAlertingEv C GC2:TermConnCreatedEv TermC GC2: TermConnRingingEv TermC GC2: CallCtlTermConnRingingEvTermC: GC2 CallCtlConnEstablishedEv C GC2 CallCtlTermConnTalkingEv termC CiscoTransferStartEv GC1 ConnCreatedEv C GC1 CiscoHuntConnectionCreatedEv B GC1: ConnConnectedEv B GC1: CallCtlConnEstablishedEv B GC1: CallCtlConnEstablishedEv C GC1:CallCtlTermConnTalkingEv TermD ... ... GC1 ConnDroppedEv D ... GC2 CallInvalidEv CiscoTransferEndEv	Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = B Call.getCurrentCallingAddress = D Call.getModifiedCalledAddress = C Call.getCurrentCallingAddress = A Call.getModifiedCalledAddress = A Call.getCurrentCalledAddress = B Call.getModifiedCalledAddress = C

Intercom

Configuration: terminal T1 has intercom line A with TargetDN B, label Bob, Unicode label UBob. Terminal T2 has intercom line B. Application provider has both T1 and T2 in control list.

C, Carol, UCarol is in the same intercom group as A, and B.

D, David, UDavid is not in the same intercom group as A, B and C.

Action	Result	Call info
Application opens provider, after provider comes in service, application issues provider.getIntercomAddresses()	JTAPI returns A and B as array of CiscoIntercomAddress.	N.A

Action	Result	Call info
Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTarget Label() request at A.	JTAPI will return B as target DN and Bob and UBob as target label.	N.A
Application issues CiscoIntercomAddress.getDafaultIntercomTargetDN(), CiscoIntercomAddress.getDefaultIntercomTargetLabel() and CiscoIntercomAddress.getDefaultIntercomUnicodeTargetLabel() request at A.	JTAPI will return B as target DN and Bob and UBob as target label.	N.A
Application issues CiscoIntercomAddres.setIntercomTarget(C, Carol, UCarol) on intercom address A. After successful response, Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A
Application1 is observing CiscoIntercomAddress A and has AddressObserverAdded to it. Application2 sets intercom target, label to C, Carol, UCarol.	App1 : AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL	N.A
After above step Application1 issues CiscoIntercomAddres.setIntercomTarget(B, Bob, UBob) on intercom address A.	Exception will be thrown to application as another application instance has already set the target to C, Carol, UCarol.	N.A
Intercom target DN and label for intercom address A is set to default, now application issues CiscoIntercomAddres.setIntercomTarget(D, David, UDavid) on intercom address A.	Exception will be thrown as D, David, UDavid is not in the same intercom group.	N.A
Application has set intercom target DN and label to C, Carol, UCarol for intercom address A. Now CTI Manager goes out of service, JTAPI failover to another CTIManager node. After intercom address A come back in service, JTAPI will restore intercom target DN and label to C, Carol, UCarol respectively. Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomTargetLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A

Action	Result	Call info
Application has set intercom target DN and label to C, Carol for intercom address A. Now CTI Manager goes out of service, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN, label and UnicodeLabel to C, Carol, UCarol respectively, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI.	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A
Application is connected to a CTIManager node, Cisco Unified Communications Manager node goes down, intercom device failsover to another Cisco Unified Communications Manager node, after intercom address comes back in service. CTIManager should restore intercom target Dn and label. Application issues CiscoIntercomAddress.getIntercomTargetDN(), CiscoIntercomAddress.getIntercomLabel() and CiscoIntercomAddress.getIntercomUnicodeTargetLabel() request at A.	AddressObserver at A: CiscoAddrIntercomInfoChangedEv Cause: CAUSE_NORMAL JTAPI will return C as target DN and Carol and UCarol as target label.	N.A
Application is connected to a CTIManager node, Cisco Unified Communications Manager node goes down, intercom device failsover to another Cisco Unified Communications Manager node, after intercom address comes back in service. CTIManager tries to restore intercom target Dn and label, however due to race condition some other application has already set the target Dn and Label, hence CTI is not able to restore the intercom target DN and label.	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A

Action	Result	Call info
Application is observing intercom addresses A and B. A has target set to B. User initiates intercom call. Intercom call is successful.	CallObserver at A and B: CallActiveEv GC1 Cause: CAUSE_NORMALConnCreatedEv A, Cause: CAUSE_NORMALConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMALTermConnCreatedEv A- T1 Cause: CAUSE_NORMAL TermConnActiveEv A- T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv A - T1 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL ConnCreatedEv B, Cause: CAUSE_NORMAL ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtl Cause = CAUSE_NORMAL CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
User at B presses talkback softkey to get connected to intercom initiator.	CallObserver at A and B: TermConnActiveEv B - T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CiscoRTPOutputStartedEv – T2CiscoRTPInputStartedEv – T1	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null

Action	Result	Call info
Intercom address A has target defined as B. Application initiates an intercom call by calling interface Address.ConnectIntercom() with dialeddigit as empty. Intercom call is successful.	CallObserver at A and B : CallActiveEv GC1 Cause: CAUSE_NORMAL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL TermConnCreatedEv T1 Cause: CAUSE_NORMAL TermConnActiveEv T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv T1 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL ConnCreatedEv B Cause: CAUSE_NORMAL C ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause = CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
Application initiate TerminalConnection.join() request on TerminalConnection of B to talkback. Request is successful.	CallObserver at A and B : TermConnActiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CiscoRTPOutputStartedEv – T2 CiscoRTPInputStartedEv – T1	Cg = A Cd = B CurrentCg = A CurredCd = B LRP = null
Application tried to put the intercom call on hold at A by issuing TerminalConnection.hold()	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to accept intercom call at intercom target by issuing connection.accept() at connection of B.	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to reject intercom call at intercom target by issuing connection.reject() at connection of B.	Intercom call will be disconnected.	N.A

Action	Result	Call info
Application tried to redirect intercom call by issuing connection.redirect() at connection of A or B.	PlatformException will be thrown, intercom call stay connected.	N.A
Application tried to park call by issuing connection.park() at Connection of A or B.	PlatformException will be thrown, intercom call stay connected.	N.A
Terminal T1 has intercom address A which has intercom target set to B. Terminal T2 has intercom address B and another address C. C is in call with D, GC1. A initiates intercom call to B, intercom call is auto-answered at B	No event to GC1 call, it will stay in Connected State.	N.A
Application tries to set forward on intercom address A by issuing CiscoIntercomAddress. setForwarding ()	PlatformException will be thrown.	N.A
Application tries to setRingerStatus on intercom address A by issuing CiscoIntercomAddress. setRingerStatus()	PlatformException will be thrown.	N.A
Application tries to setMessageWaiting on intercom address A by issuing CiscoIntercomAddress.setMessageWaiting()	PlatformException will be thrown.	N.A
Application tries to setAutoAcceptEnabled on intercom address A at CTIPort by issuing CiscoIntercomAddress. setAutoAcceptStatus ()	PlatformException will be thrown.	N.A
Application tries to getAutoAcceptEnabled on intercom address A at CTIPort by issuing CiscoIntercomAddress. getAutoAcceptStatus ()	PlatformException will be thrown.	N.A

DeviceState Whisper Scenario

Configuration: Terminal T1 has intercom address B, Terminal T2 has intercom address A. Application has set CiscoTermEvFilter to enable CiscoTermDeviceStateWhisperEv as well as all other DeviceState filters on T1 and T2. Application had added Terminal observer on both T1 and T2.

Action	Events	Call info
Intercom address A has target defined as B. Application initiates an intercom call by calling interface Address.ConnectIntercom() with dialeddigit as empty.	Event received at TerminalObserver of T1 CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL Event received at TerminalObserver of T2 CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_NORMAL	N.A

Action	Events	Call info
Application issue join() request on TerminalConnection of T2 (intercomTarget) to talkback to T1(intecomInitiator)	Event received at TerminalObserver of T1 None. Event received at TerminalObserver of T2 CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL	N.A
Terminal T2 already have intercom target call, Application enables CiscoTermFilter for CiscoTermDeviceStateWhisperEv.	**Event received at TerminalObserver of T2** CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_SNAPSHOT	N.A

iSac Codec

CiscoMediaTerminal Static Registration with iSac Codec

Actions	Events	Call info
1. Observe both A(CiscoMediaTerminal) and B Static Register A with media capability as CiscoMediaCapability. ISAC A calls B	CiscoTermInServirceEv for TA CiscoAddrInServiceEv for A	

Actions	Events	Call info
B answers	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoRTPOutputStartedEv for TB TA: CiscoRTPInputStartedEv for TA TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	(CiscoRTPInputStartedEv for TA).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TA).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TA).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TA).getRTPOutputProperties().getPacketSize() is not deterministic

CiscoMediaTerminal Dynamic Registration with iSac Codec

Actions	Events	Call info
1. Observe both A and B (CiscoMediaTerminal) Dynamic Register B with media capability as CiscoMediaCapability. ISAC	CiscoTermInServirceEv for TB CiscoAddrInServiceEv for B	

Actions	Events	Call info
A calls B	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB	

Actions	Events	Call info
B answers App sets RTP params on B	GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoMediaOpenLogicalChannelEv for TB TB: CiscoRTPOutputStartedEv for TA TA: CiscoRTPInputStartedEv for TB TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	CiscoMediaOpenLogicalChannelEv.getPayloadType() will return CiscoRTPPayload.ISAC CiscoMediaOpenLogicalChannelEv.getPacketSize() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPacketSize() is not deterministic

CiscoRouteTerminal Dynamic Registration with iSac Codec

Actions	Events	Call info
1. Observe both A and B (CiscoRouteTerminal) Dynamic Register B with media capability as CiscoMediaCapability. ISAC	CiscoTermInServirceEv for TB CiscoAddrInServiceEv for B	

Actions	Events	Call info
A calls B	GC1: CallActiveEv GC1: ConnCreatedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnInitiatedEv for A GC1: TermConnCreatedEv for TA GC1: TermConnActiveEvent for TA GC1: CallCtlTermConnTalkingEv for TA GC1: CallCtlConnDialingEv for A GC1: CallCtlConnEstablishedEv for B GC1: ConnCreatedEv for B GC1: ConnInProgressEv for B GC1: CallCtlConnOfferedEv for B GC1: ConnAlertingEv for B GC1: CallCtlConnAlertingEv for B GC1: TermConnCreatedEv for TB GC1: TermConnRingingEv for TB GC1: CallCtlTermConnRingingEv for TB	
B answers	GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B GC1: TermConnActiveEv for TB GC1: CallCtlTermConnTalkingEv for TB TB: CiscoMediaOpenLogicalChannelEv for TB	CiscoMediaOpenLogicalChannelEv.getPayloadType() will return CiscoRTPPayload.ISAC CiscoMediaOpenLogicalChannelEv.getPacketSize() is not deterministic

Actions	Events	Call info
App sets RTP params on B	TB: CiscoRTPOutputStartedEv for TA TA: CiscoRTPInputStartedEv for TB TA: CiscoRTPOutputStartedEv for TB TB: CiscoRTPInputStartedEv for TA	(CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getBitRate() is not deterministic (CiscoRTPInputStartedEv for TB).getRTPInputProperties().getPacketSize() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPayloadType() will return CiscoRTPPayload.ISAC (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getBitRate() is not deterministic (CiscoRTPOutputStartedEv for TB).getRTPOutputProperties().getPacketSize() is not deterministic

JTAPI Cisco Unified IP 7931G Phone Interaction

A and C are JTAPI application controllable Addresses. B1 and B2 are Address on Cisco Unified IP 7931G Terminal. Cisco Unified IP 7931G Terminal is configured to do Transfer across Addresses. B1 and B2 has shared Line B1’ and B2’ respectively configured on JTAPI controllable Terminal.

Action	Events	Call info
Scenario:1 Application is observing A: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer.	JTAPI Event received to CallObserver at A GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = null) GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1
Scenario:2 Application is observing A, B1': A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer	JTAPI Event received to CallObservers at A and B1' GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, TransferredCall = null) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
Scenario:3 Application is observing A, B1', B2': A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A and B1’ GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1’, FinalCall = GC1, TransferredCall = GC2) GC1- TermConnDroppedEv for TB1’ Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1’ Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for TB2’ Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2’ Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC-1 CiscoTransferEndEv	

Action	Events	Call info
Scenario:4 Application is observing A, B1', B2' and C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = B1 LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A, B1', B2' and C GC-1 CiscoTransferStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, TransferredCall = GC2) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause:	

Action	Events	Call info
	CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv	

Action	Events	Call info
Scenario:5 Application is observing C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = B2 Called = C CurrCalling = A CurrCalled = C LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at C GC1- CallActiveEv for callID = 101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoTransferStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER	

Action	Events	Call info
Scenario:6 Application is observing both A and C: A calls B1, B1 answers – GC1 User presses transfer key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses transfer key to complete transfer		Calling = A Called = B1 CurrCalling = A CurrCalled = C LRP = B1

Action	Events	Call info
	JTAPI events at observer of A & C: GC-1CiscoTransferStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, TransferredCall = GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER NEW META GC-1 ConnDisconnectedEv for B1 –GC1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 – GC1 Cause: CAUSE_UNKNOWN CallControlCause:	

Action	Events	Call info
	CAUSE_TRANSFER	
Scenario:7 Application is observing A: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1
Scenario:8 Application is observing A, B1': A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv TB1' GC1 CallCtlTermConnBridgedEv TB1' GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
Scenario:9 Application is observing A, B1', B2': A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference	JTAPI Event received to CallObserver at A, B1' and B2' GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv – TB1' GC1 CallCtlTermConnBridgedEv – TB1' GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoConferenceEndEv	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
Scenario:10 Application is observing A, B1', B2', and C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference		Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at A, B1', B2' and C GC-1 CiscoConferenceStartedEv (ControllerAddress = B1, ControllerTerminalConnection = TC at TB1', FinalCall = GC1, ConsultCall = GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv - TB1' GC1 CallCtlTermConnBridgedEv - TB1' GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for TC Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TC Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC-1 CiscoConferenceEndEv	

Action	Events	Call info
Scenario:11 Application is observing C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference.		Calling = B2 Called = C CurrCalling = A CurrCalled = Conference LRP = B1

Action	Events	Call info
	JTAPI Event received to CallObserver at C GC1- CallActiveEv for callID = 101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoConferenceStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for B1 Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B1 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B1 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	

Action	Events	Call info
Scenario:12 Application is observing both A and C: A calls B1, B1 answers – GC1 User presses conference key on Cisco Unified IP 7931G phone and dials C, call initiated from B2 to C: B2 calls C, C answers - GC2 User presses conference key to complete conference.	JTAPI events at observer of A & C: GC-1CiscoConferenceStartEv (ControllerAddress = B1, ControllerTerminalConnection = Null, FinalCall = GC1, ConsultCall = GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRAN CAUSE_CONFERENCE SFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	Calling = A Called = B1 CurrCalling = A CurrCalled = Conference LRP = B1

Locale Infrastructure Development Scenarios

Scenario 1—JTAPI Client Machine Has Connectivity to CallManager TFTP Server

- During install, JTAPI client would prompt user to enter TFTP IP address.
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection.
- User can select language for running JTAPI Preference application.
- JTAPI Preference application is run second time, it will present UI in the language that user selected before.

Scenario 2—JTAPI Client Machine Doesn't Have Connectivity to CallManager TFTP Server

- During install JTAPI Client would prompt user to Enter TFTP-IP Address
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection but user will have only English language to select.
- JTAPI Preference application is run second time, it will present UI in the English languages.
- TFTP connectivity is restored. Now JTAPI Preferences UI is run, it will take user to language selection

Scenario 3—JTAPI Client Machine Has Connectivity to CallManager TFTP Server

- During install JTAPI Client would prompt user to Enter TFTP-IP Address
- TFTP-IP Address is stored in JTAPI.ini parameter.
- JTAPI Preferences application is run first time, it will take user to language tab to language selection.
- User can select language for running JTAPI Preference application.
- JTAPI Preference application is run second time, it will present UI in the language that user selected before.
- Now new locale files are available with added support for a new languages.
- User runs JTAPI Preferences application, JTAPI Preferences application would notify user about available.
- Application restart JTAPI Preferences application, user will be support for new language.

Calling Party Normalization

Scenario 1—Incoming Call From a PSTN Number (Local) to JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a PSTN Number [55555555] A & the Number type is [Subscriber] through the gateway to a JTAPI Observed Terminal [2222] B.	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (55555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Subscriber

Scenario Two—Incoming Call From a National PSTN Number to JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a Dallas PSTN Number [55555555] A & the Number type is [National] through a gateway to a JTAPI Observed Terminal [2222] B.	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (97255555555) Called: B (2222) getModifiedCallingAddress (): 97255555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +197255555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: National

Scenario Three—Incoming Call From Inter-National PSTN Number to JTAPI Observed Terminal

Action	Events	Call info
A Call is offered from India PSTN Number [918028520261] & the Number type is [Inter-national] through a San Jose Gateway to a JTAPI observed Terminal [2222]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (918028520261) Called: B (2222) getModifiedCallingAddress (): 918028520261 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +918028520261 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Inter-National

Scenario Four—Outgoing Call From JTAPI Observed Terminal to PSTN Number [SUBSCRIBER]

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to a PSTN number [44444444] and the Number type is [SUBSCRIBER]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (44444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 44444444 getCurrentCalledAddress(): 44444444 getCurrentCalledPartyInfo(): 44444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType (). getNumberType () would return: Unknown.

Scenario Five—Outgoing Call From JTAPI Observed Terminal to National PSTN Number

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to a Dallas PSTN number [97244444444] & the Number type is [NATIONAL]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (97244444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 97244444444 getCurrentCalledAddress(): 97244444444 getCurrentCalledPartyInfo(): 97244444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknown.

Scenario Six—Outgoing Call From JTAPI Observed Terminal to International PSTN Number

Action	Events	Call info
A call is initiated from a JTAPI Observed Terminal 2222 through a San Jose gateway to India PSTN number [918028520261] & the Number type is [INTERNATIONAL]	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (918028520261) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 918028520261 getCurrentCalledAddress():918028520261 getCurrentCalledPartyInfo(): 918028520261 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknown.

Scenario Seven—Incoming Call From PSTN Redirected to Another PSTN by JTAPI Observed Terminal

Action	Events	Call info
A call is offered from PSTN [55555555] through a San Jose	NEW META EVENT_____META_CALL_STARTING	Calling: A (55555555)Called: B (2222)

Action	Events	Call info
Gateway to a JTAPI observed terminal [2222] which redirects the call to another San Jose PSTN [44444444]. In CallState [Idle] the fwdDestination Address (Redirect Address) should be a minus (-).	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause:CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause:CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL CallRedirectReq Redirect Address = C CallRedirectRes ConnCreatedEv at C Cause: CAUSE_REDIRECTED ConnInProgress Calling party:A, Called Party: C, LRP: B CallRedirectRes CallStateChangedEv (IDLE) Reason: REDIRECT	getModifiedCallingAddress (): 55555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: SUBSCRIBER destinationAddress: 44444444. getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Unknowns

Scenario Eight—Incoming Call From PSTN Number (Local) to JTAPI Observed Terminal Who Transfers to Another JTAPI Observed Terminal

Action	Events	Call info
A call is offered from a PSTN Number [55555555] A & the Number type is [Subscriber] through a San Jose gateway to a JTAPI observed Terminal [1111] X which transfers the call to another JTAPI Observed Terminal [2222] B	After Transfer: GC1: CiscoTransferStartEv ConnCreatedEv for B ConnConnectedEv for B CallCtlConnEstablishedEv for B TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv GC2: CiscoTransferStartEv TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv	After Transfer: Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (+140855555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfo NumberType(). getNumberType() would return: Subscriber

Click to Conference

A, B, C and D are addresses and TermA, TermB, TermC and TermD are corresponding terminals.

Action	Events	Call info
A and B are in a call GC1 created using click-to-call. User adds C to the conference call. GC2 is the initial call at C. Application is observing only C	GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: ConnInProgressEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	callingAddress = unknown calledAddress = C CurrentCalling = unknown CurrentCalled = C
	GC2: CallCtlConnOfferedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	

Action	Events	Call info

Action	Events	Call info
	GC2: ConnAlertingEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlConnAlertingEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnCreatedEv TermC GC2: TermConnRingingEv TermC CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnRingingEv TermC CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermC GC1: TermConnRingingEv TermC CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: ConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC2: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallInvalidEv CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv C CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL GC1: TermConnTalkingEv TermC CiscoFeatureReason = REASON_NORMALCause: CAUSE_NORMAL	

Action	Events	Call info
A calls B using click-to-call – GC1. A adds C to the call using click-2-conf. Application has call observer on A		Calling address: A Called address: B Current calling: A Current called: B Last redirecting party = null After C is conferenced, callinfo is not applicable.

Action	Events	Call info
	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER GC1: ConnCreatedEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnInProgressEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv A CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermA GC1: TermConnRingingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv A CiscoFeatureReason = REASON_NORMAL cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL TermConnActiveEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnTalkingEv TermA CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnInProgressEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B CiscoFeatureReason = REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv B CiscoFeatureReason = REASON_NORMAL: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL	

Action	Events	Call info
	GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	
A consults with D-GC3. A completes conference. The events received by application remains the same (as that of consult conference).	GC1: TermConnHeldEv TermA GC3: ConsultCallActiveEv GC3: ConnCreatedEv A GC3: ConnCreatedEv D GC3: CallCtlConnAlerting D GC3: ConnConnectedEv D GC3: CallCtlConnEstablishedEv B CiscoConferenceStartEv GC3->GC1 GC3: CallCtlConnDisconnectedEv A GC3: CallCtlConnDisconnectedEv D GC1: ConnCreatedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnTalkingEv TermA GC3: CallInvalidEv CiscoConferenceEndEvent	For consult call GC3: Calling address: A Called address: D Callinfo not applicable after conference is completed.
User drops D using click-2-conf feature User drops C using click-2-conference interface	GC1: ConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	Calling address: A Called address: B

Action	Events	Call info
Drop all parties a conference. A calls B using click-2-call. User adds C to the conference using click-2-conference. All parties are dropped using click to conference. Application has call observers on A, B and C.	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER GC1: ConnCreatedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	Calling address: A Called address: B GC2: Calling address = unknown Called address: C

Action	Events	Call info

Action	Events	Call info
	GC1: ConnInProgressEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnOfferedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: ConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: TermConnCreatedEv TermA GC1: TermConnRingingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnRingingEv TermA GC1: ConnConnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnActiveEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: ConnCreatedEv B Cause: CAUSE_NORMAL CiscoFeatureReason:REASON_REFER GC1: ConnInProgressEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: ConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnCreatedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

Action	Events	Call info
	GC1: TermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnActiveEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

Action	Events	Call info
	GC2: CiscoCallChangedEv GC2->GC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: ConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallInvalidEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	

Action	Events	Call info
	GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnActiveEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnTalkingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE All parties are dropped using click-2-conference GC1: TermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: TermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE	

Action	Events	Call info
	GC1: CallCtlTermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE GC1: CallInvalidEv	
A calls B using click-to-call – GC1. A adds C to the call using click-2-conf. User drops party C. Application has call observer on C only.	GC1: TermConnDroppedEv TermC CiscoFeatureReason = REASON_CONFERENCE	NA
	GC1: CallCtlTermConnDroppedEv TermC CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv A CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv A CiscoFeatureReason = REASON_CONFERENCE GC1: ConnDisconnectedEv B CiscoFeatureReason = REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv B CiscoFeatureReason = REASON_CONFERENCE GC1: CallInvalidEv	
A calls B using GC1. Address C is configured on TermC1 and TermC2. Application has call observer on C	GC2: CallActiveEv CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE	Calling = unknown Called = C Last redirecting = null
User uses click-to-conference to C.	GC2: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE	

Action	Events	Call info

Action	Events	Call info
	GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE CiscoCallChangedEv GC2->GC1 TermConn TermC1 CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason:	

Action	Events	Call info
	REASON_CLICK_TO_CONFERENCE CiscoCallChangedEv GC2->GC1 TermConn TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallInvalidEv GC1: ConnCreatedEv B Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnConnectedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	

Action	Events	Call info
	GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnPassEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnInUseEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	C answers at TermC1

Call Pickup

The basic test case for the fix was the following:

1. B and C are devices in a call pick up group. A is a device not in it.
2. A calls B.
3. C goes off-hook, and presses the Pickup softkey.
4. C is now on the call with A.

This test was run with variations in which devices were observed, and the full matrix was run. This included:

- Observing A, B, and C
- Observing A and B
- Observing A and C
- Observing B and C
- Observing only A
- Observing only B
- Observing only C

The final test run, observing only C, was the primary concern for this fix, based on customer usage. The feature request was, when only observing C, being able to get information about the original called party (A) on Pickup. All test cases passed and the correct information was displayed for all of them.

For cases 3 and 4, the call information depends on the order of events that JTAPI delivers. If JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events before the
GC2-CiscoCallChangedEv, then the call information, such as calling and called addresses, will be what was seen in
GC2. Conversely, if JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events after the
GC2-CiscoCallChangedEv, then the call information, such as calling and called addresses, will be what was seen in
GC1.

As an example, if JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events before the
GC2-CiscoCallChangedEv, the Calling Address = C, Called Address = Pickup Number. If JTAPI delivers the
GC1-CallInvalidEvent/CallObservationEndedEv events after the
GC2-CiscoCallChangedEv, the Calling Address = A, Called Address = B.

These test cases were run with auto-pickup enabled and disabled, and there was much difference in the functionality of the two. Most of the test cases are enumerated below.

The basic call from A to B is the same in all cases, and is only shown in the first case below.

Scenario One

Observing all devices and auto-pickup enabled.

Action	Events	Call info (GCID info)
A goes off-hook and dials B (Basic Call) B is ringing. C goes off-hook and presses Pickup softkey. Connection for C is dropped, B is dropped / cleaned up, C connection on Call 1 is established		Calling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B CCalling: C, CCalled: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE REASON_CALLPICKUP CCalling: A, CCalled: C LRP: NONE REASON_CALLPICKUP CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL REASON_CALLPICKUP CCalling: A, CCalled: C REASON_NORMAL

Action	Events	Call info (GCID info)
	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv	

Action	Events	Call info (GCID info)
	GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallCtlConnEstablishedEv-C	

Note Both B and C in the following scenarios have exactly the same behavior and events. Only the behavior of device C (the one picking up the call) changes.

Scenario Two

Observing all devices with auto-pickup disabled.

Action	Events	Call ID Info
C goes off-hook and presses Pickup softkey	GC2-CallActiveEvent	CCalling C, CCalled: NONE
Call 2 gets dropped or invalidated	GC2-ConnCreatedEvent-C	LRP: NONE
C gets a connection on Call 1	GC2-ConnConnectedEvent-C	REASON_NORMAL
B is dropped from Call 1	GC2-CallCtlConnInitiatedEv-C	REASON_CALLPICKUP
C is ringing	GC2-TermConnCreatedEvent	CCalling A, CCalled: C
C is on call with A	GC2-TermConnActiveEvent	Calling: A, Called: C, LRP: B
	GC2-CallCtlTermConnTalkingEv	REASON_CALLPICKUP
		Calling A, CCalled: C
	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	Calling: A, Called: C, LRP: B REASON_CALLPICKUP REASON_NORMAL REASON_NORMAL
	GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C	
	GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	
	GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	
	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	

The flow of events differs greatly when the auto-pickup option is enabled or disabled. When Auto Call Pickup is disabled and a user presses the Pickup softkey (C), the phone rings. The user has to answer the phone as if it is a normal call. When the phone is ringing, the original call that was created when they went offhook is terminated, they are connected to the existing call, and the old party (B) is removed from the call. There is no CiscoCallChangedEv generated when Auto Call Pickup is disabled, because the call does not change, it is terminated before C joins the new call.

A Group Pickup scenario follows, during which the Group Pickup softkey is used in place of the Pickup softkey. This required actually dialing the number for the pickup group. Group Pickup also is subject to the Auto Call Pickup service parameter. The general flow and call events are identical to the normal Call Pickup scenarios, except with added events for the required dialing of the pickup number.

Scenario Three

Observing all devices with group pickup and auto-pickup enabled.

Action	Call event	Call ID Info
C goes offhook and presses Group Pickup softkey C is dialing the PU Number C is added to the original call Pickup added to original call Pickup # is removed Call 2C is dropped from Call 2Pickup # is removed Call 1 B is dropped / invalidated	GC1 [add to others to clarify] GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv **GC2-CallCtlConnDialingEv-C** **GC2-ConnCreatedEvent-PU** **GC2-ConnInprogressEvent-PU** **GC2-CallCtlConnEstablishedEv-C** GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C **GC1-ConnCreatedEvent-PU** GC1-ConnConnectedEvent-C **GC1-CallCtlConnEstablishedEv-C** GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv **GC1-ConnInprogressEvent-PU** **GC1-CallCtlConnOfferedEv-PU** **GC2-ConnDisconnectedEvent-PU** **GC2-CallCtlConnDisconnectedEv-PU** GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP:B REASON_CALLPICKUP, LRP: PU CCalling: C, CCalled: PU REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP

There are only a handful of changes for the above Group Pickup case, and they all directly relate to the extra required step of dialing the pickup number.

Scenario Four

Observing all devices with Group Pickup and Auto-Pickup disabled.

Action	Event	Call info
C goes offhook and pressed “Group Pickup” softkey C is dialing the PU number PU is removed from Call 2 C is removed from Call 2 Call 2 is destroyed C gets a connection on Call 1 B is dropped from Call 1 C is ringingC picks up	GC1 GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent[ADDRS] GC1-ConnInprogressEvent GC1-CallCtlConnOfferedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent GC1-CallCtlConnDisconnectedEv GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

The tables above have scenarios during which all of the devices were observed. The devices were run with every possible combination, across all varieties of Pickup and Group Pickup. Parts of the scenarios had the exact same output and others were redundant and are not shown here. For example, device A and B were identical and shown only once.

Scenario Five

Only observing device B.

Action	Call events	Call IDs/Call info
A is in the process of calling B B is ringing A is removed from Call 1 B is removed from Call 1	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallInvalidEvent GC1-CallObservationEndedEv	CCalling: A, CCalled: B, Calling: A, Called: B, LRP: NONE REASON_NORMAL REASON_CALLPICKUP REASON_NORMAL

Scenario Six

Observing only device A.

Action	Call events	Call IDs/Call info
A goes offhook and dials B B is ringing C is ringing B is removed from Call 1	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C	CCalling: A, CCalled: NO, NO LRP REASON_NORMAL CCalling A, CCalled B, Called: NOT SET LRP: NONE CCalling: A, CCalled: C, LRP: B Called: NOT SET REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_NORMAL

Scenario Seven

Observing only device C with Auto-Pickup enabled.

Action	Call events	Call IDs/Call info
C goes offhook and presses “Pickup” hotkey C is connected to Call 1 C is dropped from Call 2Call 2 is invalidated / cleared A and C are connected on Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-CallCtlConnEstablishedEv-C	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REAON_CALLPICKUP CCalling A, CCalled: NONE LRP: NONE CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_CALLPICKUP REASON_CALLPICKUP CCalling A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

Scenario Eight

Observing only device C with Auto-Pickup disabled.

Action	Call events	Call IDs/Call info
C goes offhook and pressed “Pickup” softkey Call 2 is destroyed C is added to Call 1, but does not pick upC is ringing C picks up, and is connected to Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LR PREASON_NORMAL REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Scenario Nine

Observing only device C with Group Pickup and AutoPickup enabled.

Action	Call event	Call IDs/Call info
C goes offhook and presses “Pickup” softkey C dials the Pickup Number C is added to Call 1PU is added to Call 1 PU # is removed from Call 2 C is removed from Call 2Call 2 I invalidated / cleared C is connected to Call 1 PU is removed from Call 1	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-PU GC1-CallCtlConnEstablishedEv-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PUCCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP CCalling: A, C Called: C CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKU PREASON_NORMAL CCalling: A, CCalled: C REASON_CALLPICKUP CCalling: A, CCalled: C REASON_CALLPICKUP

Scenario Ten

Observing only device C with Group Pickup and Auto-Pickup disabled.

Action	Call events	Call IDs/Call info
C goes offhook, and presses “Group Pickup” softkey C dials the PU Number PU is dropped from Call 2C is dropped from Call 2Call 2 is destroyed C is added to Call 1 C is ringing C is connected to A	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC1-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_CALLPICKUP REASON_NOTMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

selectRoute() with Calling Search Space and Feature Priority

The selectRoute() API with calling search space and feature priority as array of int. is shown in the following table.

Action	Events	Call info
Add call observer on phones A, B, C, D. Register Route Point RP Register route call back, with select route API with three rows route selected: A, ...CSS: 0, FP: 1 route selected: B, ...CSS: 1, FP:3 route selected: D, ...CSS: 1, FP:1 C calls RP Call rings at A. A answers. C-A call is connected.	GC1 CallActiveEv GC1 ConnCreatedEv C: GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C: GC1 TermConnCreatedEv TC GC1 TermConnActiveEv TC GC1 CallCtlTermConnTalkingEv TC GC1 CallCtlConnDialingEv C: GC1 CallCtlConnEstablishedEv C: GC1 ConnCreatedEv RP: GC1 ConnInProgressEv RP: GC1 CallCtlConnOfferedEv RP: After redirect request is processed GC1 ConnCreatedEv A: GC1 ConnInProgressEv A: GC1 CallCtlConnOfferedEv A: GC1 ConnDisconnectedEv RP: GC1 CallCtlConnDisconnectedEv RP: GC1 ConnAlertingEv A: GC1 CallCtlConnAlertingEv A: GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnConnectedEv A: GC1 CallCtlConnEstablishedEv A: GC1 TermConnActiveEv A GC1 TermConnActiveEv A [C] CiscoRTPInputStartedEv [A] CiscoRTPOutputStartedEv [A] CiscoRTPInputStartedEv [C] CiscoRTPOutputStartedEv	calling: C lastRedirected:RP called: A

Extension Mobility Login Username

Terminal A is in control list of user, Terminal B is not in control list of User. Extension Mobility login username is John, end user id user for application is John.

Action	Result	Call info
Open provider, Terminal A doesn't have any observer, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A.	InvalidStateException is thrown.	NA
Open provider, Add Observer to Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A.	Application should get empty string “” for username.	NA
Open provider, User “John” EMLogin to Terminal A and add observer to the Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal A, now open provider, add observer to Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal A, now open provider, add observer to Terminal A, User “John” EMLogout of Terminal A, Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get empty string “” for username	NA

Action	Result	Call info
OpenProvider, User “John” EMLogin to Terminal B, add observer, application calls CiscoTerminal.getEMLoginUserName() at Terminal B **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA
User “John” EMLogin to Terminal B, OpenProvider, add observer to Terminal B, Application calls CiscoTerminal.getEMLoginUserName() at Terminal B **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String “John”	NA

Terminal A is in control list of user and configured with the Extension Mobility logout profile of user Kerry. The Kerry profile is configured with logout username as Kerry. There is another profile with login username of John.

Action	Result	Call info
User John logs into Terminal A, OpenProvider and add observer to Terminal A. Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. John logs out at Terminal A. Application calls CiscoTerminal.getEMLoginUserName() at Terminal A. **Note** Application verifies if EM login has been done by invoking CiscoTerminal.getLoginType().	Application should get String John. Application should get String Kerry.	NA NA

Calling Party IP Address

The following are some examples of call scenarios.

Basic Call Scenario

- JTAPI application monitors party B
- Party A is an IP phone
- A calls B

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

Consultation Transfer Scenario

- JTAPI application monitors party C
- Party B is an IP phone
- A talking B
- B initiates a consultation transfer call to C
- IP Address of B is available to JTAPI application monitoring party C.

Consultation Conference Scenario

- JTAPI application monitors party C
- Party B is an IP phone
- A talking B
- B initiates a consultation conference call to C
- IP Address of B is available to JTAPI application monitoring party C.

Redirect Scenario

- JTAPI application monitors party B and party C
- Party A is an IP phone
- A calls B
- IP Address of A is available to JTAPI application monitoring party B
- Party A redirects B to party C (
- Calling IP address is not available to JTAPI application monitoring party B (not a supported scenario).
- Calling IP address of B is provided to JTAPI application monitoring party C.

CiscoJtapiProperties

1. Set Socket Connect Timeout to 5 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager and do a normal provider open. Expected Result: Socket Connect to Primary CTI Manager should fail in not more than 5 secs
2. Set Socket Connect Timeout to 5 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager, set security options to True and do a secured provider open. Expected Result: Socket Connect to Primary CTI Manager should fail in not more than 5 secs (Socket Connect timed-out in ~5 seconds, though it took some additional time initially for verifying security certificates)
3. Set Socket Connect Timeout to 0 seconds; Plug out the Ethernet cable for PRIMARY CTI Manager, set security options to true and do a secured provider open. Expected Result: Socket Connect to Primary CTI

Manager will no longer rely on new Service Parameter (Socket Connect timed-out in ~23 seconds, though it took some additional time initially for verifying security certificates).

IPv6 Support

Use Case1 - Basic Call Scenario: Calling Is IPv6 Enabled Phone; Called Is IPv6

Action	Events	Call info/Expected result
IPv6 enabled phone A calls JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoOutputStartedEv will contain the far-end Ipv6 RTP address(of A) getRemoteAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone(B)

Action	Events	Call info/Expected result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case2 - Basic Call Scenario: Calling Is IPv6 Enabled Phone; Called Is IPv4

Action	Events	Call info/Expected result
IPv6 enabled phone A calls JTAPI Observed IPv4 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv4 RTP address of the monitored phone

Action	Events	Call info/Expected result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case3 - Basic Call Scenario: Calling Is IPv4 Enabled Phone; Called Is IPv6

Action	Events	Call info/Expected Result
IPv4 enabled phone A calls JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object. CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case4 - Basic Call Scenario: Calling Is IPv4 Enabled Phone; Called Is IPv4

Action	Events	Call info/Expected Result
IPv4 enabled phone A calls JTAPI Observed IPv4 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object. CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv4 RTP address. getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv4 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case5 - Consultation Transfer Scenario, IPv6 Device Consults

Action	Events	Call info/Expected Result
GC1: Call between A & B Consult Call: IPv6 enabled phone B consults JTAPI Observed device C for Transfer using GC2.	NEW META EVENT_____META_CALL_STARTING	For Consult Call: CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for B in an InetAddress object to the JTAPI Application observing C. While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return null
C Answers	**CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case6 - Consultation Transfer Scenario, IPv4 Device Consults

Action	Events	Call info/Expected Result
GC1: Call between A & B Consult Call: IPv4 enabled phone B consults JTAPI Observed device C for Transfer using GC2.	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for B in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null
C Answers	**CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case7: Redirect Scenario

Action	Events	Call info/Expected Result
GC1: Call between A(IPv6) & B Redirect Call: phone B redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return IPv6 format address for A in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return null

Action	Events	Call info/Expected Result
C Answers		

Action	Events	Call info/Expected Result
	CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause :	

Action	Events	Call info/Expected Result
	CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case8: Redirect Scenario (IPv4)

Action	Events	Call info/Expected results
GC1: Call between A(IPv4) & B Redirect Call: phone B redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for A in an InetAddress object to the JTAPI Application observing C While, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() will return null

Action	Events	Call info/Expected results
C Answers		

Action	Events	Call info/Expected results
	CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause :	

Action	Events	Call info/Expected results
	CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case9: Redirect Scenario, Calling Device Redirects

Action	Events	Call info/Expected Result
GC1: A calls B(IPv4) Redirect Call: Phone A redirects call to JTAPI Observed device C using GC2.	New Meta Event_____META_CALL_STARTING **CallActiveEv** for callID = GC2 Cause: CAUSE_NEW_CALL	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() will return IPv4 format address for B in an InetAddress object to the JTAPI Application observing C CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() or, CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr()_v6 **will not return** IP address for A in an InetAddress object to the JTAPI Application observing B after redirect.

Action	Events	Call info/Expected Result
C Answers		

Action	Events	Call info/Expected Result
	ConnCreatedEv for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for A Cause : CAUSE_NORMAL **ConnDisconnectedEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for A Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL	

Action	Events	Call info/Expected Result
	TermConnActiveEv for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case10: Route Scenario, IPv6 Enabled Calls RoutePoint Which Routes Call to IPv6 Device

Action	Events	Call info/Expected Result
IPv6 enabled phone A calls RoutePoint which routes the call to JTAPI Observed IPv6 enabled device B using GC1.	NEW META EVENT_____META_CALL_STARTING	CiscoRouteEvent.getCallingPartyIpAddr_v6() will return IPv6 format address for A as an InetAddress object. While, CiscoRouteEvent. getCallingPartyIpAddr() will return null getRemoteAddress() on CiscoRTPOutputProperties in CiscoRTPOutputStartedEv will contain the far-end Ipv6 RTP address getLocalAddress() on CiscoRTPInputProperties in CiscoRTPInputStartedEv will contain the Ipv6 RTP address of the monitored phone

Action	Events	Call info/Expected Result
B Answers	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallRouteEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

Use Case11: Enterprise Parameter "Enable IPv6" is Enabled

Application does an open provider by providing the list of CTI Manager IPs as

- IPv4 address of CTI Manager1
- IPv6 address of CTI Manager1
- IPv4 address of CTI Manager2
- IPv6 address of CTI Manager2

Now once the JTAPI is able to establish a connection with CTI Manager and later on if CTI Manager1 goes down, in failover attempt application can see delay in connecting as JTAPI will first try to connect with IPv6

address of CTI Manager1 (which is next in the list) even though that IP address is of the same CTI Manager and only once it times out it will try with the IPv4 address of the CTI Manager2 which will succeed (assuming CTI Manager2 is running).

Provider Open Scenario

1. Service Parameter for Reconnect Attempt is not set(or set to 0), Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is stared again and connection is restored.
   - Enterprise parameter “Enable IPv6”is enabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address.
2. Service Parameter for Reconnect Attempt is not set (or set to 0), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is stared again and connection is restored.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address. But, the existing devices registered with IPv6 address will be closed with “CiscoTermRegistrationFailedEv” with a new reason code “IP_CAPABILITY_MISMATCH”
3. Service Parameter for Reconnect Attempt is not set (or set to 0), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv6 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager indefinitely till the CTI Manager is started again and connection is restored.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will not be able to reconnect to CTI Manager, as it no longer supports IPv6 address but JTAPI will try reconnecting to CTI Manager indefinitely till the time service parameter is again enabled and CTI Service restarted.
4. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
   - Enterprise parameter “Enable IPv6”is enabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address.
5. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv4 address. JTAPI will be able to open a connection with CTI Manager.

- CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
- Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will be able to reconnect to CTI Manager with the same IPv4 address. But, the existing devices registered with IPv6 address will be closed with “CiscoTermRegistrationFailedEv” with a new reason code “IP_CAPABILITY_MISMATCH”

6. Service Parameter for Reconnect Attempt is set to some integer value (say 5), Enterprise parameter “Enable IPv6” is enabled. Application tries to open a provider with IPv6 address. JTAPI will be able to open a connection with CTI Manager.
   - CTI Manager is stopped – JTAPI will try reconnecting to CTI manager 5 times before closing all the opened devices and provider.
   - Enterprise parameter “Enable IPv6”is disabled and CTI manager is restarted – JTAPI will not be able to reconnect to CTI Manager, as it no longer supports IPv6 address but JTAPI will try reconnecting to CTI Manager 5 more times (as the same can again be enabled on Cisco Unified Communications Manager) before closing all the devices and provider.
7. Enterprise parameter “Enable IPv6” is disabled. Application tries to open a provider with IPv6 address. JTAPI will not be able to open a connection with CTI manager. Retry attempts are applicable only if connection gets established once, but since in this scenario even the first attempt is failing so there will be no subsequent reconnect attempts.

   Enterprise parameter “Enable IPv6” is enabled. Application does an open provider by providing the list of CTI Manager IPs as
   - IPv4 address of CTI Manager1
   - IPv6 address of CTI Manager1
   - IPv4 address of CTI Manager2
   - IPv6 address of CTI Manager2

   Now once the JTAPI is able to establish a connection with CTI Manager and later on if CTI Manager1 goes down, in failover attempt application can see delay in connecting as JTAPI will first try to connect with IPv6 address of CTI Manager1 (which is next in the list) even though that IP address is of the same CTI Manager and only onMangerce it times out it will try with the IPv4 address of the CTI Manager2 which will succeed (assuming CTI Manager2 is running).

Calling Party IP Address Scenarios

1. Ipv6 enabled phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv6 calling party IP address.

   getCallingPartyIpAddr() will return NULL

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address.
2. Ipv4 enabled phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 calling party IP address (existing behavior)

getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

getCallingPartyIpAddr_v6() will return NULL.

3. Ipv6 only phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv6 calling party IP address.

   getCallingPartyIpAddr() will return NULL

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

4. Ipv4 enabled phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 formatted calling party IP address.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return NULL.

5. Ipv4_v6(Two Stack) phone calls a CTI controllable device. Subsequently, the CTI controllable device is monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 and Ipv6 calling party IP addresses.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

6. Ipv4_v6(Two Stack) phone calls a CTI controllable device that is already monitored by a JTAPI application. JTAPI will generate a CiscoCallCtlConnOfferedEv (non-Route Points) or CiscoRouteEvent (Route Points) notification containing an Ipv4 and Ipv6 calling party IP addresses.

   getCallingPartyIpAddr() will return the actual calling Party IPv4 address.

   getCallingPartyIpAddr_v6() will return the actual calling Party IPv6 address

RTP Addresses

1. An Ipv6 enabled phone calls an Ipv6 JTAPI Observed phone and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.

2. An Ipv4 enabled phone calls an Ipv4 JTAPI Observed phone and the call is answered. JTAPI will generate(existing behavior):
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.

3. An Ipv4 enabled phone calls an Ipv6 JTAPI Observed device and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.

4. An Ipv6 enabled phone calls an Ipv4 JTAPI Observed device and the call is answered. JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
5. A Dual stack(Ipv4_v6) phone calls another dual stack(Ipv4_v6) JTAPI Observed device, preferred media termination is set to IPv6, and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
6. A Dual stack(Ipv4_v6) phone calls another dual stack(Ipv4_v6) JTAPI Observed device, preferred media termination is set to IPv4, and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
7. A Dual stack(Ipv4_v6) phone calls an Ipv4 JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
8. A Dual stack(Ipv4_v6) phone calls an Ipv6 JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
9. An IPv4 phone calls a dual stack (Ipv4_v6) JTAPI Observed device and the call is answered then JTAPI will generate:
   - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the calling device.
   - CiscoRTPInputStartedEv containing the Ipv4 RTP address of the monitored phone.
10. An IPv6 phone calls a dual stack (Ipv4_v6) JTAPI Observed device and the call is answered then JTAPI will generate:
    - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of the calling device.
    - CiscoRTPInputStartedEv containing the Ipv6 RTP address of the monitored phone.
11. JTAPI observed IPv6 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for Transfer. C answers and B completes the Transfer, then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of C.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of A.

- At C:
  - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address of A.
  - CiscoRTPInputStartedEv containing the Ipv4 RTP address of C.

12. JTAPI observed IPv4 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for Transfer. C answers and B completes the Transfer, then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv4 RTP address of A.
    - At C:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of C.
13. JTAPI observed IPv6 phone(A) calls JTAPI observed IPv4 phone(B). B answers and consults IPv6 phone(C) for conference. C answers and B completes the conference. Conference Bridge has an IPv4 address. Then JTAPI will generate:
    - At A:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of A.
    - At B:
      - CiscoRTPOutputStartedEv containing the far-end Ipv4 RTP address of the Conference Bridge.
      - CiscoRTPInputStartedEv containing the Ipv4 RTP address of B.
    - At C:
      - CiscoRTPOutputStartedEv containing the far-end Ipv6 RTP address which corresponds to the MTP that was automatically inserted by Call Manager to perform Ipv4/Ipv6 conversion.
      - CiscoRTPInputStartedEv containing the Ipv6 RTP address of C.

CTI Port/Route Point Registration Scenarios

1. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv6 address and Application addressing capability as IPv6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv6 address.

2. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 address and Application addressing capability as IPv4. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 address.
3. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 and IPv6 addresses and Application addressing capability as IPv4_v6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 and IPv6 addresses.
4. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv4 address and Application addressing capability as IPv4. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv4 address.
5. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only’. Application tries to do a static register of that CTI Port/Route Point to CTIManager with IPv6 address and Application addressing capability as IPv6. The registration will succeed and CTI Port/Route Point will get registered with CTIManager with IPv6 address.
6. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only’. Application tries a static register of that CTI Port/Route Point by providing an IPv6 address or/and by advertising application addressing capability as IPv6 (or Ipv4_v6) only then request will fail with a CiscoRegistrationException.
7. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4 (or IPv4_v6) only. Then the request will be denied with a CiscoRegistrationException.
8. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4 only (or IPv4_v6 both)’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4 only. Then the registration will succeed and CTI Port/Route point will get registered with IPv4 address when the same is provided with SetRTPParams request.
9. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv6 only (or IPv4_v6 both)’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv6 only. Then the registration will succeed and CTI Port/Route point will get registered with IPv6 address when the same is provided with SetRTPParams request.
10. CTI Port/Route Point has ‘IP Addressing Mode’ configured as ‘IPv4_v6 both’. Application tries to dynamically register that CTI Port/Route Point to CTIManager. IP capabilities advertised by the application at the time of registration are IPv4_v6 both. Then the registration will succeed and CTI Port/Route point will get registered with IPv4_v6 address when the same is provided with SetRTPParams request.
11. If an application tries to dynamically register a CTI Port/Route Point by advertising its IP capabilities as IPv6, which is already registered to another application with IPv4 address. Then the request will be declined with a CiscoRegistrationException or “CiscoTermRegistrationFailedEv” will be sent with new reason code “IP_CAPABILITY_MISMATCH”.

Advance Test Cases

1. Application does a provider Open with IPv4 address to a CTI Manager which has enterprise parameter “Enable IPv6” enabled. Application tries to register a CTI Port/Route point with an IPv6 address whose device IP Addressing Mode is set to “IPv4_v6” by advertising applications addressing capability as “IPv6 only”. The registration request will succeed.
2. JTAPI observed IPv6 device A calls another JTAPI observed IPv4 device B, call is offered and answered at B. In that case:

   CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() will return NULL

   CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() will the actual calling Party IPv6 address

   - At B:
     - CiscoRTPInputStartedEv will have B’s IPv4 Address
     - CiscoRTPOutputStartedEv will have IPv4 address of the MTP Resource

   Interesting thing to notice here is CiscoRTPOutputStartedEv has an IPv4 address while calling party IP Address is an IPv6 address.

Direct Transfer Across Lines Use Cases

Action	Events	Call info/Expected Result
Application is observing A, B1, B2, and C (B1 and B2 are two Addresses on the same Terminal TB) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing A, B1, B2, and C (B1 and B2 are two Addresses on the same Terminal which allows connected transfer across lines over phone manually which supports this feature) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2		CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
User B2 presses transfer and user selects active call(A–> B call) from the phone UI and presses transfer again to do connected Transfer Across Lines	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B2 GC3: ConnConnectedEv for B2 GC3: CallCtlConnInitiatedEv for B2 GC3: TermConnCreatedEv for TB2 GC3: TermConnActiveEvent for TB2 GC3: CallCtlTermConnTalkingEv for TB2	

Action	Events	Call info/Expected Result
User Presses transfer again to complete connected transfer across lines	GC3: TermConnDroppedEv for TB2 GC3: CallCtlTermConnDroppedEv for TB2 GC3: ConnDisconnectedEv for B2 GC3: CallCtlConnDisconnectedEv for B2 GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Action	Events	Call info/Expected Result
Application is observing A, B1, B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
Application is observing B1, B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEv GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEvGC1: CallObservationEndedEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing only B1: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	JTAPI will throw PlatformException “Transfer controller is not set and could not find a suitable TerminalConnection”. Since JTAPI cannot get/find call leg for B2 from GC2	

Action	Events	Call info/Expected Result
Application is observing only A: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 User presses transfer and selects active call(A –> B call) from the phone UI and presses Transfer again to do Connected Transfer Across Lines	GC2: CallActiveEvent GC2: ConnCreatedEv for A GC2: ConnCreatedEv for C GC1: CiscoCallChangedEv GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for A GC2: TermConnActiveEvent for A GC2: CallCtlTermConnTalkingEv for A GC2: ConnConnectedEv for C GC2: CallCtlConnEstablishedEv for C GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: TermConnDroppedEv for A GC1: CallCtlTermConnDroppedEv for A GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: CallInvalidEvent GC1: CallObservationEndedEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2
Application is observing only B2: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 setTransferController to B1 GC1.transfer(GC2)	JTAPI will throw PlatformException “Transfer controller is not set and could not find a suitable TerminalConnection” Since JTAPI cannot get/find call leg for B1 from GC1	
Application is observing only C: A calls B1, B1 answers – GC1 B2 calls C, C answers - GC2 User presses transfer and selects active call(A–>B call) from the phone UI and preses Transfer again to do Connected Transfer Across Lines.	GC2: CiscoTransferStartEv GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for AGC2: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
New user role(Standard Supports Connected Xfer/Conf) is associated with application user Application opens a provider and disassociates the above mentioned user role	JTAPI delivers: ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoTermRestrictedEv CiscoAddrRestrictedEv (for all the phones that support connected tx/conf across lines)	CiscoProviderCapabilityChangedEv. hasConnectedTransfer ConferenceCapabilityChanged() returns True

Action	Events	Call info/Expected Result
Application is observing A, B1, B2, C and C' (B1 and B2 are two Addresses on the same Terminal, C' is sharedline of C) A calls B1, B1 answers – GC1 B2 calls C, C answers – GC2 setTransferController to B1 GC1.transfer(GC2)	At Transfer: GC1: CiscoTransferStartEvGC2: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC' GC1: TermConnPassiveEvent for TC' GC1: CallCtlTermConnInUseEv for TC' GC2: TermConnDroppedEv for TC' GC2: CallCtlTermConnDroppedEv for TC' GC2: CiscoCallChangedEv GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv. getControllerTerminalName() returns Terminal name for B1&B2

Action	Events	Call info/Expected Result
New user role(Standard Supports Connected Xfer/Conf) is not associated with application user Application tries to add observer on phone which allows connected transfer/conference across lines	Phones that allow connected transfer/conference across lines are exposed as restricted. JTAPI throws PlatformExceptionImpl ("Terminal is restricted", CiscoJtapiException.CTIERR_DEVICE_RESTRICTED).	CiscoTerminal.isRestricted() returns TRUE
New user role(Standard Supports Connected Xfer/Conf) is not associated with application user Application opens a provider and associates the above mentioned user role	JTAPI delivers: ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoAddrActivatedEv CiscoTermActivatedEv (for all the phones that support connected tx/conf across lines)	CiscoProviderCapabilityChangedEv. hasConnectedTransferConference CapabilityChanged() returns True

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

Action	Events	Call info/Expected result
New Role “Standard Supports Connected Xfer/Conf” to control phones which support connected conference across lines is Not Associated with user.Phones TA(Line A), TB(Lines B1, B2) and T3(Lines C); TC is a phones which allows connected conference across lines. Observe All; GC1: A calls B1, GC2: B2 calls C		

Action	Events	Call info/Expected result
Do connected Conference Across Lines manually on Phone TB (which supports this feature) to conference GC1 and GC2	App, gets an PlatformExceptionImpl ("Terminal is restricted", CiscoJtapiException.CTIERR_DEVICE_RESTRICTED) when the add observer on TB, B1 and B2 GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	CiscoConferenceStartEv.getControllerAddress() returns B1CiscoConferenceStartEv.getControllerTerminalName() returns TB

Enhanced MWI Use Cases

Action	Result
Application calls CiscoAddress.setMessageSummary() to set voice and fax counts on a phone that supports the enhanced message waiting counts.	Phone displays updated voice and fax counts provided and also updates the MWI indicator accordingly. A successful response is returned.
Application calls CiscoAddress.setMessageSummary() to set voice and fax counts on a phone that does not support the enhanced message waiting counts.	Phone only updates the MWI indicator accordingly—no voice and fax counts are displayed on the phone. A successful response is returned
Application calls CiscoAddress.setMessageSummary() to set voice counts, but the “high priority” voice counts provided are bigger than “total” voice counts provided.	The request fails with the following error returned: INVALID_HIGH_PRIORITY_VOICE_COUNTS

Action	Result
Application calls CiscoAddress.setMessageSummary() to set fax counts, but the “total” fax counts provided is bigger than maximum size allowed.	The request fails with the following error returned: INVALID_TOTAL_FAX_COUNTS

Join Across Lines Enhancements

A, C, D, E and F are addresses on different terminals. B1 and B2 are addresses on the same terminal TermB.

A, B1 and C are in a conference call GC1 with B1 as the controller and connected to conference bridge Conference-1. B2, D and E are in conference call GC2 with D as controller and connected to bridge Conference-2.

Action	Events
Application conferences the two calls on B1 and B2 by invoking GC1.conference(GC2) to chain two conference call.	**Events to CallObserver of A, C and B1:** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2Ev.getConferenceChain().getChainedConferenceCalls() will returnGC1 **Event for CallObserver at B2, D & E:** ConnDisconnectedEv B2 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection of Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 & Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 & GC2

Action	Events
Application invokes GC2.conference(GC1) to chain two conference calls.	**Event for CallObserver at B2, D & E:** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 **Events for CallObservers at A, B1 & C:** ConnDisconnectedEv B1 GC1 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will returnGC1

Action	Events
A, B1, C are in conference-1 (GC1), B1, D, E are in conference-2 (GC2), B2, F, G are in conference-3 (GC-3) Application completes conference at C by initiating GC1.conference(GC2, GC3) setting B1 as controller.	

Action	Events
	Event for CallObserver at A, B1 & C: TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-2 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 Ev.getConferenceChain().getChainedConferenceCalls() will return GC1 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEvTermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection will return connection for Conference-3 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-2 & Conference-3 Ev.getConferenceChain().getChainedConferenceCalls() will return GC2 & GC3 **Event for CallObserver at B1, D & E:** ConnDisconnectedEv B1 GC2 Cause = NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1-GC2 Ev.getConferenceChain().getChainedConferenceCalls() will returnGC2 **Event for CallObserver at B2, F & G:**

Action	Events
	ConnDisconnectedEv B2 GC3 Cause = NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause = NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause = NORMAL, callCtlCause = CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC3 Ev.getAddedConnection will return connection for Conference-1 Ev.getConferenceChain().getChainedConferenceConnections() will return connections of Conference-1 Ev.getConferenceChain().getChainedConferenceCalls() will return GC3

Call Scenario: A, B1 and C are in conference call GC1 with B1 as controller. B2 is in call GC2 with D

Action	Events
Application sets the requestor as B2 and calls GC2.conference(GC1) getControllerAddress() returns B2. getOriginalControllerAddress() returns B1.	A CiscoConferenceStartEv CallCtlTermConnTalkingEvTermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
If application uses B1 as request controller in the above setup getControllerAddress() returns B1. getOriginalControllerAddress() returns B1.	Events are same as above

Swap or Cancel and Transfer or Conference Behavior Change

Use Case 1		
Connected Transfer on the phone which allows connected Transfer	GC1 & GC2 call will be created as normal.	
A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2	GC1: CallCtlTermConnHeldEv for TB	
User B presses transfer and user selects active call(A–>B call) from the phone UI	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
User B presses transfer again	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Use case 2		
Connected Transfer on phone with sharedline (A and A' are sharedlines) A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer and selects active calls (A–>B call), User B presses transfer again	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB| GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA' GC2: TermConnPassiveEvent for TA' GC2: CallCtlTermConnInUseEv for TA' GC1: TermConnDroppedEv for TA' GC1: CallCtlTermConnDroppedEv for TA' GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2:TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

Use case 3		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer hard key User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns GC3

Use case 4a		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer hard key User press select active calls key. User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns null

Use case 4b		
Connected Transfer/Conference – Cancel feature A calls B, B answers – GC1 B puts A–>B call on hold B calls C, C answers – GC2 User B presses transfer (or conference) hard key. User press select active calls key and also selects GC1 (A‡B call) User B presses cancel key	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelled Ev.getConsultCall() returns GC1

Use case 5		
Consult Transfer – Swap calls A calls B, B answers – GC1 B puts A–>B call on hold B setup consult Transfer to C, C answers – GC2 User B presses Swap key, User B presses transfer to complete the transfer	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL

Use case 6		
Consult Transfer – Swap/Cancel A calls B, B answers – GC1 A puts A–>B call on hold B setup consult Transfer to C, C answers – GC2 User B presses press Swap softkey, User B presses Cancel softkey	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnHeldEv for TB For TA (GC2), CallCtlTermConnHeldEv For TA (GC1), CallCtlTermConnTalkingEv GC1: CiscoCallFeatureCancelledEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 7		
Consultative Transfer Initiated from Phone, App sends SetupTransfer/Conference request – request fails A calls B, B answers – GC1 B setups transfer call to C B calls C, C answers – GC2 Application creates a new call and sends another consult() request	GC1 & GC2 call will be created as normal. Request will fail with PlatformException “CTIERR_CONSULTCALL_ALREADY_OUTSTANDING”	

Use case 8a		
Consult Transfer/Conference – Application Resumes primary call on phone which supports connected transfer/conference and sends another consult setup request GC1: A calls B GC2: B consults C Application resumes GC1 on TB Application creates another call and sends consult() request to call D; D answers	GC1 and GC2 will be created as normal For TB (GC2), CallCtlTermConnHeldEv For TB (GC1), CallCtlTermConnTalkingEv CiscoCallFeatureCancelledEv Consult call will go through and GC3 will be created as normal	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 8b		
Consult Transfer/Conference – Manually Resume primary call on phone which supports connected transfer/conference and then sends another consult setup request GC1: A calls B GC2: B consults C User manually resumes (SWAP) GC1 on B Application creates another call and sends consult() request to call D; D answers	GC1 and GC2 will be created as normal On Manual Resume or Swap, Consult Call will not be cancelled on the phone, nor will application get CiscoCallFeatureCancelledEv. When application tries to setup another consult, it will go through (GC3 will be created as normal) and it will cancel the existing consult call and application will get: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getCall() returns GC1 CiscoCallFeatureCancelledEv.getConsultCall() returns GC2

Use case 9		
Connected ConferenceA (Phone which allows connected conference) calls B, B answer, B puts A onhold, B calls C, C answer B press “Conference” hardkey, and picks active call from UI, and selects A‡B call B press “Conference” again to complete connected conference	GC1 & GC2 call will be created as normal. C1: CallCtlTermConnHeldEv for TB GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	

Use case 10		
Consult Conference from Phone, then Swap and complete conference through phone A calls B, B answerB setup conference to C, C answer B press “Swap” softkey A press “Conference”	GC1 & GC2 call will be created as normal. GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL

Use case 11		
Consult Conference from Phone and then Swap and Cancel conference thru phone A calls B, B answer B setup conference to C, C answer A press “Swap” key, and picks active call from UI, and selects A–>B call B press “Cancel”	GC1 & GC2 call will be created as normal. GC1: CallCtlTermConnTalkingEv for TB GC2: CallCtlTermConnHeldEv for TB GC1: CiscoCallFeatureCancelledEv(consultCall = GC2)	getCiscoFeatureReason() returns CiscoFeatureReason. REASON_NORMAL CiscoCallFeatureCancelled Ev.getCall() returns GC1 CiscoCallFeatureCancelled Ev.getConsultCall() returns GC2

Use case 12		
Connected Conference Across Lines	Same as JAL scenario but we will have a temporary call GC3	

Use case 13		
Manual Consult followed by transfer complete by application GC1: A calls B1 GC2: B1 setups consult call to C manually over phone G1.transfer(GC2)	At Transfer: GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC1: CiscoCallFeatureCancelledEv GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEvent for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEv	

Use case 14		
Manual consult followed by conference complete by application GC1: A calls B1 GC2: B1 setups consult call to C manually over phone G1.conference(GC2)	At Conference: **GC1: CiscoCallFeatureCancelledEv** GC1: CiscoConferenceStartEv GC1: CiscoCallFeatureCancelledEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoConferenceEndEv	

Drop Any Party Use Cases

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 1 Application is observing A, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of C. Application invokes Connection.disconnect() on Connection of A.	InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 2 Application is observing C, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case3 Application is observing A and C. B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 4 Application is observing B, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv And B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 5 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE.
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 6 Application is observing A, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of C. Application invokes Connection.disconnect() on Connection of A.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 7 Application is observing C, B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 8 Application is observing A and C. B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C.	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Use Case 9 Application is observing B, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 10 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE. A and A’ are shared line
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	Call info
Use Case 11 Application is observing A, B is conference controller. A, A' and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getPartyInfo() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) is dropped out of conference TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are A(abc) is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 12 Application is observing A', B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv . A'("xyz") is dropped out of conference... InvalidStateException is thrown. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'("xyz") is dropped out of conference..	N, A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 13 Application is observing A and A'. B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" InvalidStateException is thrown. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A'(xyz) is dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) is dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) and A'(xyz)is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	Call info
Use Case 14 Application is observing B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" No Events A'(xyz) is dropped out of conference. No Events A(abc) is dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEv B is disconnected from conference.g ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	Call info
Use Case 15 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A(xyz), dropped out of conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc), dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is disconnected from conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.

- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE. A and A’ are shared line
- Cisco Unified Communications Manager service parameter “Drop Ad Hoc Conference” set “never”

Scenario	Action	Result	CallInfo
Use Case 16 Application is observing A, B is conference controller. A, A’ and B are in conference. Displayname for A is “abc”, and displayname for A’ is “xyz”	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of “abc” and “xyz” ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. No Events. A’(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc) is dropped out of conference.	N.A Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 17 Application is observing A', B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. No Events. A(abc) is disconnected from conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'(xyz) dropped out of conference TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(xyz) is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 18 Application is observing A and A'. B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' No Events. A'(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(xyz) is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

Scenario	Action	Result	CallInfo
Use Case 19 Application is observing B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect() on Connection of B. Application invokes Connection.disconnect(xyz) on Connection of A. Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid B is disconnected from conference. No Events. A'(xyz) is disconnected from conference. No Events A(abc) dropped out of conference ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
Use Case 20 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Display name for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames () at Connection of A. Application invokes Connection.disconnect () on Connection of B Application invokes Connection.disconnect (xyz) on Connection of A. Application invokes Connection.disconnect (abc) on Connection of A. Application invokes Connection.disconnect() on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B is dropped out of conference. TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A'(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

A, B, C are in conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns "True"	N..A
A, B, C are in conference. B drops from conference	Application invokes CiscoCall.isConferenceCall()	Interface Returns "False"	N..A

A, B, B’ are in conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A
A, B, B’ are in conference, B’ drops from conference.	Application invokes CiscoCall.isConferenceCall()	Interface Returns “False”	N..A
A, B, C are in conference. Applications opens provider, gets snapshot call event	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A
A, B, B’ are in conference. Applications opens provider, gets snapshot call event	Application invokes CiscoCall.isConferenceCall()	Interface Returns “True”	N..A

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 21 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEV A, B C is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to TRUE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 22 Application is observing A, B and C, and B is conference controller. A, B, and C are in conference.	Application invokes Connection.disconnect() on Connection of A. Application invokes Connection.disconnect() on Connection of C Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A is dropped out of conference. TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C is dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, B, and C are dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter “Advanced Ad Hoc Conference Enable” is set to FALSE.
- Cisco Unified Communications Manager service parameter “**Drop Ad Hoc Conference”** set “When controller leaves”

Scenario	Action	Result	CallInfo
Use Case 23 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference. Displayname for A is "abc", and displayname for A' is "xyz"	Application invokes Connection.getDisplayNames() at Connection of A. Application invokes Connection.disconnect(xyz) on Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz" TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' A(xyz), dropped out of conference.	N.A. Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	Application invokes Connection.disconnect(abc) on Connection of A. Application invokes Connection.disconnect() on Connection of B.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc), dropped out of conference. TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, A' and B is disconnected from conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

Scenario	Action	Result	CallInfo
	Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA' CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv Connections of A, and B are disconnected, A(abc) and A'(xyz) will be dropped, and since only B is left, it will also get dropped and call goes Invalid.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI parameter is enabled to allow dropAnyPartyFeature.
- Cisco Unified Communications Manager service parameter "Advanced Ad Hoc Conference Enable" is set to TRUE.
- Cisco Unified Communications Manager service parameter "**Drop Ad Hoc Conference"** set "When controller leaves"

Scenario	Action	Result	CallInfo
Use Case 24 Application is observing A, A' and B, and B is conference controller. A, A', and B are in conference.	Application invokes Connection.getDisplayNames() at Connection of A.	JTAPI returns CiscoPartyInfo[] with CiscoPartyInfo.getDisplayName() of "abc" and "xyz"	N.A.

Scenario	Action	Result	CallInfo
Display name for A is “abc”, and displayname for A’ is “xyz”	Application invokes Connection.disconnect () on Connection of B Application invokes Connection.disconnect (xyz) on Connection of A. Application invokes Connection.disconnect (abc) on Connection of A.	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A, A’ and B is dropped out of conference. TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ A’(xyz) is disconnected from conference. TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A(abc) dropped out of conference	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
Application invokes Connection.disconnect() on Connection of A.	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA’ ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A(abc), A(xyz) and B is dropped out of conference.	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL	

Park Monitoring Support

Phone B—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Phone A—Future models.

Phone A'—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Park DN—P1, P2

Phone C—Cisco Unified IP Phone 7900 Series with SIP/SCCP

All the default values for the Park Monitoring Reversion timer and Park Monitoring Forward No reversion timers apply.

Use Case 1: Park Monitoring States

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 2 After step 1, Park Monitoring reversion timer expires after the configured time	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3 After step 1 or 2, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = RETRIEVED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 4 After step 1 or 2 above, B drops off the call invoking CiscoConnection.disconnect() on the connection of B.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = ABANDONED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 5 Consider Park Monitoring forward no retrieve destination on A is configured as F After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to F	**Events received at Call Observer on A, B** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 6 Consider Forward no retrieve destination on A is configured to self • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 7 Consider Forward no retrieve destination is not configured • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 2: Shared Line Scenario - Cisco Unified IP Phone Does Park

Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Consider Forward no retrieve destination is not configured, • Consider Park Monitoring Reversion timer and Park Monitoring Forward no reversion timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A **Note** all shared lines ring as is today	Reason = CiscoFeatureReason.PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 3: Shared Line Scenario - Cisco Unified IP Phone 7900 Series with SIP Does Park

Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A' answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Note** New event is not seen as Cisco Unified IP Phone 7900 Series does park	

Action	Result	Event/Call info
Step 2 Consider Park Reversion timer expires • Call is reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Note** All shared lines including the Cisco Unified IP Phone (future model) phone A receives the incoming call	Reason = CiscoFeatureReason.PARKREMINDER

Use Case 4: Use Case for Snap Shot Scenario

Initial scenario: Application has added Call Observer on A, B. Application has NOT added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 After step 1, application now adds Address Observer on A.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 3a After step 2, consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3b After step 1, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
Step 4 After step 3, application now adds Address Observer on A.	**New address event with park** state = RETRIEVED is not received at A, since the call is already retrieved	

Use Case 5: Park DN Is Monitored

Initial scenario: Application has added Call Observer on A, B. Application invokes registerFeature() API on Provider in order to monitor park DN P1. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Provider Observer Prov1** CiscoProvCallParkEv **Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 6: Query Number of Parked Calls

Initial scenario: Application has added Call Observer on A, B, C.

Action	Result	Event/Call info
Step 1 B calls A. A answers. Application invokes CiscoConnection.park() on connection on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 C calls A. A answers. Application invokes CiscoConnection.park() on connection on A for the second call on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
Step 3 Application invokes CiscoAddress.getAddress CallInfo(Term A) Application invokes CiscoAddress CallInfo.getNumParkedCalls()	CiscoAddressCallInfo is returned which includes information about number of parked calls getNumParkedCalls() returns 2	

Use Case 7: Filter Enabling or Disabling

Initial scenario: Application has added Call Observer on A, B. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Initially filter is disabled. • Application adds AddressObserver on A. • Application now invokes CiscoConnection.park() on connection on A. • Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event received as filter is disabled	

Action	Result	Event/Call info
Step 2 After step 1, Application enables filter via setCiscoAddrPark StatusEvFilter(true) and then by invoking CiscoAddress. setFilter(CiscoAddrEvFilter), for being able to receive the events.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 8: Filter Enabling or Disabling

Initial scenario: From the phone B calls A. A answers.(Call Observers are not added)

Action	Result	Event/Call info
Step 1 Initially filter is enabled. • Application adds AddressObserver on A. • Application now invokes park directly from the phone A. • Park reversion timer starts	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 9: Filter Enabling or Disabling

Initial scenario: From the phone B calls A. A answers.(Call Observers are not added)

Action	Result	Event/Call info
Step 1 Initially filter is disabled. • Application adds AddressObserve on A. • Application now invokes park directly from the phone A. • Park reversion timer starts. • Application now enables filter and invokes CiscoAddress. setFilter(CiscoAddr EvFilter)	**Events received at Address Observer on A** No event received yet, since filter is disabled **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Park reminder timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Use Case 10: Filter Enabling or Disabling

Initial Scenario : Initial scenario: Application has added Call Observer on A, B. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Initially all filters are disabled in CiscoAddEvFilter • Application adds AddressObserver on A. • Application now invokes CiscoConnection. park() on connection on A. • Park reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event received as filter is disabled	
Step 2 After step 1, Application invokes setCiscoAddrPark StatusEvFilter(true) but does not invoke CiscoAddress. setFilter (CiscoAddrEvFilter)	**Events received at Address Observer on A** No event received as the address filter is not set.	
Step 3 Now the application invokes setFilter (CiscoAddrEvFilter) on CiscoAddress	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_SNAPSSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Additional Use Cases for Park Monitoring

Phone B—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Phone A—Future models.

Phone A'—Cisco Unified IP Phone 7900 Series with SIP/SCCP

Park DN—P1, P2

Phone C—Cisco Unified IP Phone 7900 Series with SIP/SCCP

All the default values for the Park Monitoring Reversion timer and Park Monitoring Forward No reversion timers apply.

1. Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 • Application invokes CiscoConnection.park() on connection on A. • Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 2 After step 1, Park Monitoring reversion timer expires after the configured time	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3 After step 1 or 2, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = RETRIEVED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 4 After step 1 or 2 above, B drops off the call invoking CiscoConnection.disconnect() on the connection of B.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = ABANDONED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 5 Consider Park Monitoring forward no retrieve destination on A is configured as F • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to F	**Events received at Call Observer on A, B** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 6 Consider Forward no retrieve destination on A is configured to self • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = FORWARD_NO_RETRIEVE Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 7 Consider Forward no retrieve destination is not configured • After step 2, Park Monitoring Forward no retrieve timer starts • Park Monitoring Forward no retrieve timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Reason = PARKREMINDER Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

2. Initial scenario: Application has added Call Observer on A, B, A'. Application has added Address Observer on A. B calls A. A/A' ring. A answers. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 Consider Forward no retrieve destination is not configured, • Consider Park Monitoring Reversion timer and Park Monitoring Forward no reversion timer expires. • Call is forwarded/reverted to parker's line A	**Events received at Call Observer on A, B** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1	
	Events received at Address Observer on A CiscoAddrParkStatusEv **A** **Note** All shared lines ring as is today	Cause = CAUSE_NORMAL park state = FORWARDED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

3. Initial scenario: Application has added Call Observer on A, B. Application has NOT added Address Observer on A. B calls A. A answers. Filter value has been set to ‘true’ through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 Application invokes CiscoConnection.park() on connection on A. Park Monitoring reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

Action	Result	Event/Call info
Step 2 After step 1, application now adds Address Observer on A.	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_SNAPSHOT park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA
Step 3a After step 2, consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** CiscoAddrParkStatusEv **A**	Cause = CAUSE_NORMAL park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 3b After step 1, application sends unpark request CiscoTerminal.unpark() on Terminal of C.	**Events received at Call Observer on A, B** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv BC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
Step 4 After step 3, application now adds Address Observer on A.	New address event with park state = RETRIEVED is not received at A, since the call is already retrieved	

4. Initial scenario: Application has added Call Observer on A, B, C. Filter value has been set to ‘true’ through setCiscoAddrParkStatusEvFilter().

Action	Result	Event/Call info
Step 1 B calls A. A answers. Application invokes CiscoConnection.park() on connection on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	
Step 2 C calls A. A answers. Application invokes CiscoConnection.park() on connection on A for the second call on A.	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
Step 3 Application invokes CiscoAddress.getAddressCallInfo(Term A) Application invokes CiscoAddressCallInfo.getNumParkedCalls()	CiscoAddressCallInfo is returned which includes information about number of parked calls getNumParkedCalls() returns 2	

5. Use case to check for address event filter to control event notification- Filter value is set to ‘false’ through setCiscoAddrParkStatusEvFilter(). This is also the default value.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 By default the address event filter value is false. Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** No event notification since filter value is false	

6. Use case to check for address event filter to control event notification. Filter value has been set to 'true' through setCiscoAddrParkStatusEvFilter().

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application enables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true). Application invokes CiscoConnection.park() on connection on A. Park Monitoring Reversion timer starts	**Events received at Call Observer on A, B** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_NORMAL park state = PARKED sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

Action	Result	Event/Call info
Step 2 After step 1 above, Application now disables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(false). Consider Park Monitoring Reversion timer expires	**Events received at Address Observer on A** No event notification as filter is disabled	
Step 3 After step 2 above, Application now enables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true).	**Events received at Address Observer on A** CiscoAddrParkStatusEv A	Cause = CAUSE_SNAPSHOT park state = REMINDER sub ID = 1234 CiscoCallID = CiscoCallID for GC1 park DN = P1 parked party = B terminal = TA

7. Use case to check the value of the filter set for the event CiscoAddrPArkrStatusEv.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers.

Action	Result	Event/Call info
Step 1 Application disables the filter through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(false) Application invokes the API getCiscoAddrParkStatusEvFilter() on CiscoAddrEvFilter.	The application receives the Boolean value ‘false’.	
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrParkStatusEvFilter(true) Application invokes the API getCiscoAddrParkStatusEvFilter on CiscoAddrEvFilter.	The Application receives the Boolean value ‘true’.	

8. Use case to check the notification of CiscoAddrIntercomInfoChangedEv and the value of the filter for the event, when the Intercom feature (target DN and/or intercom taget label) has not been changed.

   Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 Application has set the filter value to 'false' through CiscoAddrEvFilter. setAddrIntercomInfo ChangedEvFilter(false) Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value 'false'.	
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrIntercomInfo ChangedEvFilter(true) Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A No events received as the intercom Feature is unchanged. The application receives the Boolean value 'true'.	

9. Use case to check the notification of CiscoAddrIntercomInfoChangedEv and the value of the filter for the event, when the Intercom feature (target DN and/or intercom taget label) has been changed.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 Application has set the filter value to 'false' through CiscoAddrEvFilter. setAddrIntercomInfo ChangedEvFilter(false) Application issues CiscoIntercomAddress.setIntercomTarget() on intercom address A Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value 'false'.	

Action	Result	Event/Call info
Step 2 Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrIntercomInfo ChangedEvFilter(true) Application issues CiscoIntercomAddress.setIntercomTarget() on intercom address A Application invokes the API getCiscoAddrIntercomInfo ChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrIntercomInfoChangedEv **A** The application receives the Boolean value 'true'.	

10. Use case to check the notification of CiscoAddrIntercomInfoRestorationFailedEv and the value of the filter for this event.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 • Application has set the filter value to 'false' through CiscoAddrEvFilter. setCiscoAddrIntercomInfo RestorationEvFilter(false) • Application has set intercom target DN and label for intercom address A. Now CTI Manager goes outofservice, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN , label and UnicodeLabel to the set values, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI. • Applications invokes the API getCiscoAddrIntercomInfo RestorationEvFilter() on CiscoAddrEvFilter.	Events Received at Address Observer on A No notification as the filter is disabled. The Application receives a Boolean value 'false'	

Action	Result	Event/Call info
Step 2 • The application enables the filter through the API CiscoAddrEvFilter. setCiscoAddrIntercomInfo RestorationEvFilter(true) • Application has set intercom target DN and label for intercom address A. Now CTI Manager goes outofservice, JTAPI failsover to another CTIManager node. After intercom address A come back in service, JTAPI tries to restore intercom target DN , label and UnicodeLabel to the set values, however due to race condition some other application has already set the target DN, JTAPI get failure response from CTI. • Applications invokes the API getCiscoAddrIntercomInfo RestorationEvFilter() on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrIntercomInfoRestorationFailedEv **A** The Application receives a Boolean value ‘true’	

11. Use case to check the notification of CiscoAddrRecordingConfigChangedEv and the value of the filter for this event.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Recording Profile Configurations Settings have not been changed

Action	Result	Event/Call info
Step 1 • Application has set the filter value to ‘false’ through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter is set to default value (false) • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value ‘false’.	

Action	Result	Event/Call info
Step 2 • Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter(true) • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A No events as the Recording settings are unchanged. The application receives the Boolean value ‘true’.	

12. Use case to check the notification of CiscoAddrRecordingConfigChangedEv and the value of the filter for this event.

Initial scenario: Application has added Call Observer on A and B. Application has added Address Observer on A. B calls A. A answers

Action	Result	Event/Call info
Step 1 • Application has set the filter value to ‘false’ through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter (false)User changes the Recording Profile Configurations, through the Admin Pages. • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter() on CiscoAddrEvFilter.	**Events received at Address Observer on A** No address notification as filter is disabled. The application receives the Boolean value ‘false’.	
Step 2 • Application enables the filter value through CiscoAddrEvFilter. setCiscoAddrRecording ConfigChangedEvFilter(true)User changes the Recording Profile Configurations, through the Admin Pages. • Application invokes the API getCiscoAddrRecording ConfigChangedEvFilter on CiscoAddrEvFilter.	Events Received at Address Observer on A CiscoAddrRecordingConfigChangedEv **A** The application receives the Boolean value ‘true’.	

Use Cases Related to DPark

Initial set up:

-Application has added call observer on B and A

-User has configured DPark DN **D**

- B is a future model Cisco Unified IP Phone

-A calls B. B answers with GCID GC1

Call Scenario	Expected behavior
Assisted DPark from a Cisco Unified IP Phone: • Cisco Unified IP Phone phone B (which is on active call with A) presses the pre-configured 'DPark BLF' button • The parked party A will be connected to D and hearMoH	When A(parked party) is connected to D, the following events are received **Events received at Call Observer on B, A** GC1 CallCtlTermConnHeldEv TB (CiscoFeatureReason. REASON_DPARK_CALLPARK) GC1 CiscoTermConnSelectChangedEv TB GC1 ConnUnknownEv B GC1 CallCtlConnUnknownEv B GC1 TermConnUnknownEv TB (CiscoFeatureReason. REASON_REFER)) GC1 ConnCreatedEv DGC1 ConnInProgressEv DGC1 CallCtlConnQueuedEv D (CiscoFeatureReason. REASON_REFER)) GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TBGC1 ConnDisconnectedEv BGC1 CallCtlConnDisconnectedEv B(CiscoFeatureReason. REASON_REFER))
DPark from Cisco Unified IP Phone • Cisco Unified IP Phone phone B (which is on active call with A) presses the 'Transfer' softkey • Parked party A is put on hold, and parker B dials **D** • Parker B is connected to D and hears MOH. • Parker B presses"Transfer" softkey again to complete the transfer of the parked party A to Dpark code D. • Parked party A is connected to D	No change in behavior. All events/reason remain the same as is today
DPark from JTAPI API • Application requests for a consult call from B, using the consult() API with destination address as DPark DN D. Say this call has GCID–GC2 • Application complete transfer, using the transfer() API GC1.transfer(GC2) • When transfer is completed A is connected to DPark DN	No change in behavior. All events/reason remain the same as is today

Call Scenario	Expected behavior
Unpark from JTAPI API • Consider application is observing C. • After step 3, application issues a request to unpark using the connect() API, with destination address as the prefix code followed by DPark code. • A is connected to C	No change in behavior. All events/reason remain the same as is today
Redirect to DPark DN via JTAPI API • B redirects to DPark code D, via the redirect() API with redirect destination as D.	No change in behavior. B is connected to DPark DN, but no park operation.

Logical Partitioning Feature Use Cases

Redirect from a Logical Partition (LP) Restricted Cluster

Terminal TA is configured with address A and is registered to a cluster which is configured with logical partition restrictions. Terminal TX is registered with address X which is configured to a cluster with no LP configuration.

Action	Result
X calls A. A redirects the call to a local PSTN number	PlatformException is thrown to redirect request. getErrorCode() on the exception returns CiscoJtapiException. REDIRECT_CALL_PARTITIONING_POLICY

Action	Result
A calls X (GC1) , X redirects the call to its local PSTN number	Originating cluster recognizes that the call is redirect to a PSTN and disconnects the call **Events delivered to Call Observer of A** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnDisconnected X GC1 CallCtlConnDisconnectedEv X GC1 CallInvalidEv

Call forward: Call to a address which is forward all to PSTM in GeoLocation with “disallowed” policy

Action	Result
setForward on A t o local PSTN Application is monitoring X. • X calls A using GC1.connect()	Connect() API succeeds but the call is dropped due to restrictions on A side. **Events delivered to call observer of X** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv X GC1 CallCtlTermConnDroppedEv TX GC1 ConnDisconnectedEv X GC1 CallCtlConnDisconnectedEv X GC1 ConnDisconnected A GC1 CallCtlConnDisconnectedEv A GC1 CallInvalidEv

Call Transfer: Transferring a call from different geo location to PSTN by controller in GeoLocation with “disallowed” policy

Action	Result
X calls A, A consults to PSTN number. Application is monitoring A. • A completes the transfer.	Platform exception is thrown to transfer() request. getErrorCode() returns CiscoJtapiException. TRANSFERFAILED

Call Conference: Conferencing a call from different location to PSTN by controller in GeoLocation with "disallowed" policy

Action	Result
X calls A, A consults to PSTN number. Application is monitoring A. • A completes the conference.	Platform exception is thrown to conference() request. getErrorCode() returns CiscoJtapiException. CTIERR_FEATURE_NOT_AVAILABLE.

Call Park / UnPark: Parking and un parking a PSTN call.

A and B are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as B

Action	Result
PSTN number calls A, A answers and parks the call. Application is monitoring A and B • B un-parks the call using unpark() API.	Call fails: ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED

Shared Lines

TermA and TermA' are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as TermA.

Action	Result
PSTN number calls A. Only TermA rings	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

Call Park Reversion with Shared Lines in Different Geographic Locations

TermA and TermA' are in the same cluster but configured in different geo locations with LP restrictions. PSTN is the same geo location as TermA.

Action	Result
PSTN number calls A, TermA answers and parks the call. After time out call is offered at TermA and not at TermA'	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

ComponentUpdater Enhancement Use Cases

Action	Result
Application calls ComponentUpdater(null)	Updater.log is creaated in the same directory
Application calls ComponentUpdater("C:\\temp\\")	Updater.log is created in c:\temp
Application calls ComponentUpdater(“readonlydirectory”) Application does not have write permission to Readonlydirectory	No log is created.

IPv6 Support

Use case for getIPAddressingMode()

Action	Result
Step 1 IP Adrressing mode for terminal A in Cisco Unified Communications Manager Admin pages is set as IPv4 Application invokes CiscoTerminal.getIPAddressingMode() on terminal A.	getIPAddressingMode() returns 0
Step 2 After step 1, the IP Addressing mode is changed from IPv4 to IPv6. User would be prompted to re set the device. Now application invokes CiscoTerminal.getIPAddressingMode() on terminal A.	getIPAddressingMode() returns 1(provided user had re-set the device)

Support for Cisco Unified IP Phone 6900 Series

Scenario / Description	Events to application
Application connects to CTIManager. • TermA is a Cisco Unified IP Phone 6921. • TermB is a Cisco Unified IP Phone 7931 with roll over mode. Admin now enables the new user role.	CiscoProviderCapabilityChangedEv – CiscoProvider.canObserverTerminalsWithRoleOver() returns true. CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() returns true. **Events to Provider Observer:** CiscoTermActivatedEv TermA CiscoTermA ctivatedEv TermB

Scenario / Description	Events to application
Admin removes the new user role.	CiscoProviderCapabilityChangedEv – CiscoProvider.canObserverlTerminalsWithRoleOver() returns false. CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() returns true. **Events to Provider Observer:** CiscoTermRestrictedEv TermACiscoTermRestrictedEv TermB
Term A is a Cisco Unfied IP 6900 series phone. Application does not have the new role enabled. Term A is in application control list Application adds observer on TermA	PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_DEVICE_RESTRICTED
Call Scenarios:	
Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1) Application issues consult request to Y (GC2). Call is created on A:P1	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEvGC2: ConnCreatedEv **A:P1** GC2: ConnConnectedEv **A:P1** GC2: CallCtlConnInitiatedEv **A:P1**
No roll over for incoming calls: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1). Applications calls A from B using connect API.	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv **B** GC2: ConnConnectedEv **B** GC2: CallCtlConnInitiatedEv **B** GC2: TermConnCreatedEv TernB GC2: CallCtlConnDialingEv B GC2: CallCtlConnEstablishedEv B GC2: ConnFailedEv B. getCiscoCause() returns CiscoCallEv.CAUSE_USERBUSY

Scenario / Description	Events to application
Roll over for Transfer and Conference (consult())only: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on terminal. X calls A, application answers the call (GC1). Applications calls connect() API from Adress A to Y. Similar exception will be seen for unPark(), startMonitor() requests.	Events delivered to Terminal Observer GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED
Only 1 address has callObserver: Term A is configured with adress A, A:P1, A:P2 where P1 and P2 are 2 partitions. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to same DN with max calls and busy trigger set to 1. Application adds callObserver on address A only. X calls A, call is answered Applications consults with Y using consult() API. On phone call consult call is created on A:P1	**Events delivered to CallObserver on A** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException is thrown. **getErrorDescription()** returns (“No callobserver on address A:P1). getErrorCode() returns **CiscoJtapiException.** ASSOCIATED_LINE_NOT_OPEN
Roll over to any line: In roll over, preference is giving to addresses with the same DN. If an address with the same DN is available it is choosen to roll over the consult call. Term A is configured with adress A, **B**, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1. Application adds callObserver on TermA X calls A, application answers the call. Applicaton consults with Y. The consult call is created on line3.	**Events delivered to Terminal Observer** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv A:P1 GC2: ConnConnectedEv A:P1 GC2: CallCtlConnInitiatedEv A:P1

Scenario / Description	Events to application
Roll over to any line (same DN has another call): Term A is configured with adress A, **B**, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1. Application adds callObserver on TermA GC1: A:P1 calls Z, A:P1 holds the call GC2:X calls A, application answers the call Application consults GC2 to Y (GC3) Application completes the transfer	**Events delivered to Terminal Observer** GC2: ConnConnectedEv **A** GC2: CallCtlConnEstablishedEv **A** GC2: CallCtlTermConnTalkingEv TermA GC2: CallCtlTermConnHeldEv TermA GC3: CallActiveEv GC3: ConnCreatedEv B GC3: ConnConnectedEv B GC3: CallCtlConnInitiatedEv B … GC3: ConnCreatedEv Y .. GC3: CallCtlConnAlertingEv Y .. GC3: ConnConnectedEv Y GC3: CallCtlConnEstablishedEv Y CiscoTransferStartEv getTransferControllerAddress() returns A….CiscoTransferEndEv
Max call > 1: Term A is configured with adress A, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 3 and 2 on A. Application adds callObserver on TermA. X call A, A answers Application consults with Y Consult is setup on A (same line)	**Events delivered to Terminal Observer** … … GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv **A** GC2: ConnConnectedEv **A** GC2: CallCtlConnInitiatedEv **A**
Term A is configured with address A, A:P1 where P1 is partition. Application has the new role “Standard CTI Allow Control of Phones supporting roll over mode”enabled. Term A is configured to roll over calls to any line with max calls and busy trigger set to 1/1 on A and A:P1. Application adds callObserver on TermA. A1 calls X. X answers the call – GC1 Y calls A. A answers the call and calls consult to Z	PlatformException is thrown. getErrorCode() returns CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED

Terminal and Address Capability Settings Use Cases

Call Scenario	Expected behavior
Max Calls, busy trigger and Line Button: Address A is configured on TermA and TermA1. Line on TermA is configured with max calls 2 and line on TermA1 is configured with max calls 3. A on TermA is configured with busy trigger of 1 and A on TermA1 is configured with busy trigger of 2. A is on button 1 on TermA and on button 2 on TermA1.	A.getMaxCalls(TermA) returns 2 A.getMaxCalls(TermA1) returns 3 A.getMaxCalls(null) return 2 or 3 A.getBusyTrigger(TermA) returns 1 A.getBusyTrigger(TermA1) returns 2 A.getButtonPosition(TermA) returns 1 A.getButtonPosition(TermA1) returns 2
Voice Mail Pilot: Voice mail profile on address A is configured to point to pilot 2001 Voice mail profile on A is changed to point to to pilot 2002	A.getVoiceMailPilot() returns 2001 = CiscoAddrVoiceMailPilotChangedEv Ev.getAddress.getVoiceMailPilot() returns 2002
Labels: Address B on TermB is configured with ascii label = assciB and unicode label unicodeB.	B.getAsciiLabel(null) returns “asciiB” B.getUnicodeLabel(null) returns “unicodeB” B.getAsciiLabel(TermB) returns “asciiB” B.getUnicodeLabel(TermB) returns “unicodeB” B.getAsciiLabel(TermC) – throws Exception
IP Address TermC is registered in IPV4 mode only TermD is registered in IPV6 mode only TermE is registered in dual mode	TermC.getIPV4Address() returns a valid InetAddress TermC.getIPV6Address() returns null TermD.getIPV6Address() returns a valid InetAddress TermE.getIPV4Address() returns a valid InetAddress TermE.getIPV6Address() returns a valid InetAddress

Call Scenario	Expected behavior
Features: DTAL: TermF is Cisco Unified IP Phone model 7970 TermG is a CiscoRouteTerminal TermH is a CiscoMediaTerminal Join Across Lines: TermI has join across lines option enabled ConsultCallRollOver: TermJ is Cisco Unified IP Phone model 6921	TermF.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns true; TermF.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns true; TermG.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns false; TermG.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns false; TermH.canDirectTransferAcrossLines(CiscoTerminal .APPLICATION) returns true; TermH.canDirectTransferAcrossLines(CiscoTerminal .PHONE_USER) returns false; TermI. canJoinAcrossLines (CiscoTerminal .APPLICATION) returns true; TermI. canJoinAcrossLines (CiscoTerminal .PHONE_USER) returns true; TermJ canJ ConsultCallRollOver (CiscoTerminal .APPLICATION) returns false; TermJ can ConsultCallRollOver (CiscoTerminal .PHONE_USER) returns false;
Provider has term1 and term2 in control list and both are registered to CUCM Application gets provider and registers for the register and unregister events. Provider.registerFeature(CiscoProvFeatureID. TERMINAL_REGISTER_UNREGISTER_EVENT_NOTIFY Term1 unregisters	Provider Observer will get CiscoProvTermialUnRegisteredEv – getTerminal() returns Term1. Term1.isRegistered() returns false.
Term1 registers	CiscoProvTermialRegisteredEv – getTerminal() returns Term1. Term1.isRegistered() returns true.
Roll Over: TermK is Cisco Unified IP Phone model 7931 configured with rollover to any line. TermL is Cisco Unified IP Phone model 7940	TermK.getRollOverConfig() returns CiscoTerminal.ROLLOVER_ANY_DN. TermL.getRollOverConfig() returnd CiscoTerminal.NO_ROLLOVER.

All of the following use cases use the same basic configuration, unless specifically noted in the use case itsellf:

Pickup group1: N:P1 (pickup group number = N, partition = P1)

Pickup group2: M:P1

A, B and C are defined to be in pickup group1

D, E, and F are defined to be in pickup group2

Pickup group2 is subgroup for pickup group1

The following scenarios are basic use cases for the new Call Pickup APIs, and do not require an enumerated event list because of their simplicity. After these, two Call Pickup use cases will be presented so that you can see the new events in place. Interested parties should refer to the Unison JTAPI Interface Specification (EDCS-614242) for the full set of Pickup Use Cases. The Use Cases in that document will not have the new event, but after reading these use cases it should be readily apparent where they belong in the existing use cases.

Scenario	Action	Result
JTAPI Application is observing C	Application opens provider and issues CiscoAddress.getPickupGroupDN() and CiscoAddress.getPickupGroupPartition() at C	API returns N and P1 respectively
Cisco UCM service parameter AutoCallPickupEnabled is set to false.	Application opens provider and issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup()	API returns FALSE
Cisco UCM service parameter AutoCallPickupEnabled is set to true.	Application issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup()	API returns TRUE
Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	Application opens provider and issues Provider.getCapabilities() and then calls API CiscoProviderCapabilities.canAutoPickup() ii). Now adminitrator sets AutoCallPickupEnabled service parameter to TRUE, iii). Application calls API CiscoProviderCapabilities.canAutoPickup()	i) API returns FALSE ii). CiscoProviderCapabilityChangedEv is delivered iii). API returns TRUE
JTAPI Application is observing C.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application opens provider and issues provider.unregisterPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. iv). A calls B	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Unregistration successful. iv). No Events for pickup alert.

Scenario	Action	Result
JTAPI Application is observing C. Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls starts ringing at C.
JTAPI Application is observing C. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call is answered at C.
JTAPI Application is observing C and D Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.groupPickup(Address of D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call starts ringing at D.
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTermina;.groupPickup(Address of D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call is answered at D.
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to FALSE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii).Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B call starts ringing at D.

Scenario	Action	Result
JTAPI Application is observing C and D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i)Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). Application issues CiscoTerminal.otherPickup(Address of C) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at D
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B and then E calls C iii). Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 CiscoProvPickupCallAlertEvent with calling E, Called C, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at D. (Longest ringing call is picked up)
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.pickup(Address of C) at C	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Application will get PlatformException with error CTIERR_NO_CALLS_TO_PICKUP.
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.groupPickup(Address fo D, N) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Request successful, new call created, but call gets disconnected with reason NORMAL.

Scenario	Action	Result
JTAPI Application is observing C, D. Cisco UCM service parameter AutoCallPickupEnabled is set to TRUE.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). A calls B iii). A-B call goes IDLE. Then Application issues CiscoTerminal.otherPickup(Address of D) at D	i) Registration successful. ii). After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). Application will get PlatformException with error CTIERR_NO_CALLS_TO_PICKUP and cause NORMAL
JTAPI Application is observing C, D.	i). Application opens provider and issues provider. registerPickupAlert(K, P1) to register for pickup alert notification for call pickup group K:P1, where K is not a valid group number	i) PlatformException thrown with error CTIERR_INVALID_GROUP_NUMBER and cause NORMAL
	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii). Application again issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1.	i). Registration successful ii.) Registration blocked at the JTAPI layer, and a InvalidStateException is thrown to the application with error CTIERR_ALREADY_REGISTERED and description “Pickup Group already registerred / observed”.
	i). Application opens provider and issues provider.deregisterPickupAlert(N, P1) to deregister for pickup alert notification for call pickup group N:P1, which was not registerred for previously.	i). InvalidStateException thrown to application with error CTIERR_REGISTRATION_NOT_FOUND and description, “Pickup group is not registerred with this provider”.
JTAPI Application is observing J. Cisco UCM service parameter “AutoCallPickupEnabled” is set to true. J is an address in two partitions, P1 and P2. J belongs to the Pickup Group with number N.	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.pickup(Address of J in P1) at terminal of J in P1	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at J in P1.

Scenario	Action	Result
JTAPI Application is observing J. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. J is an address in two partitions, P1 and P2. J belongs to the Pickup Group with number N..	i). Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.pickup(Address of J in P2) at terminal of J in P2	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii). A-B calls answered at J in P2.
JTAPI Application is observing K. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. K is an address in partition P3, and belongs to Pickup Group 3, in partiton P3. Pickup Group 3 has the same number as Pickup Group 1, N. CSS of K is configured to check P3:P1.	i) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B. iii.) Application issues CiscoTerminal.groupPickup(Address of K in P3, N) at terminal of K in P3.	i). Registration successful ii.) After Calll Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii.) InvalidStateException thrown to application with error CTIERR_NO_CALLS_TO_PICKUPand description, "There are no calls to pickup".
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) The CUCM crashes / goes offline, and CTI failsover to the the secondary CUCM server.	i). Registration successful ii.) The JTAPI layer sense the failover and re-registers for the pickup groups that it was registers for (N, P1).
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) An administrator logs into the CUCM Admin page and changes the DN for the Pickup Group.i).	i) Registration successful ii.) Application recieves a ProviderPickupNotificationRegistrationClosedEvent, with reason = CTIERR_PICKUPGROUP_CHANGED
	i.) Application opens provider and issues provider. registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) An administratior logs in to the CUCM Admin page and changes the "Auto Pickup Enabled" service parameter.	i.) Registration successful ii.) Application receives a CiscoProviderCapabilityChangedEv. CiscoProviderCapabilities. canAutoPickup() will be changed to reflect the new setting.

Scenario	Action	Result
JTAPI Application is observing J. Cisco UCM service parameter "AutoCallPickupEnabled" is set to true. J is an address in partition P3. J belongs to the Pickup Group with number N, in P1. J's CSS is not configured to check P1.	i) Application opens provider and issues provider.registerPickupAlert(N, P1) to register for pickup alert notification for call pickup group N:P1. ii.) A calls B iii.) Application issues CiscoTerminal.pickup(Address of J in P3) at terminal of J in P3.	i) Registration successful ii.) After Call Pickup Group notification time, applications gets CiscoProvPickupCallAlertEvent with calling A, Called B, pickup group number : N, pickup group partition: P1 iii.) PlatformException is thrown with reason = CTIERR_TEMPORARY_FAILURE and description "Temporary Failure".

Full-Event Use Case 1: (Observing All Devices): Auto-Pickup Disabled

Action	Call event	Call ID / Info
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes Terminal.pickup(Address of C)	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv CiscoProvPickupCallAlertEvent GC2-CallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B Calling A, Called B, PUG Number N, PUG partition P1 CCalling C, CCalled: NONE LRP: NONE REASON_NORMAL

Action	Call event	Call ID / Info
Call 2 gets dropped / invalidated C gets a connection on Call 1 B is dropped from Call 1 C is ringing C is on call with A	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	REASON_CALLPICKUP CCalling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP CCalling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP REASON_NORMAL REASON_NORMAL

Full-Event Use Case 2 (Observing All Devices): Auto-Pickup Enabled

Actions	Events	Call info (GCID: Info)
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing … pause for Pickup Group's delay Provider recieves Pickup Alert C invokes Terminal.pickup(Address of C)	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE

Actions	Events	Call info (GCID: Info)
Old Conn for C is dropped B is dropped / cleaned up C's connection on Call 1 is established	GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallCtlConnEstablishedEv-C	REASON_CALLPICKUP CCalling: A, CCalled: C LRP: NONE REASON_CALLPICKUP CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL REASON_CALLPICKUP CCalling: A, CCalled: C LRP: B REASON_NORMAL

Full-Event Use Case 3 (Observing All Devices): Group Pickup, Auto-Pickup Enabled

Action	Call event	Call Id/Info
provider. registerPickupAlert(N, P1) [Basic call happens, see case 1] … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes groupPickup(Address of C, N) C is dialing the PU Number C is added to the original call Pickup added to original call	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP:B

Action	Call event	Call Id/Info
Pickup # is removed Call 2 C is dropped from Call 2 Pickup # is removed Call 1 B is dropped / invalidated	GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	REASON_CALLPICKUP, LRP: PU CCalling: C, CCalled: PU REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP

As you can see, there are only a handful of changes for this Group Pickup case, and they all directly relate to the extra required step of dialing the Pickup Number. A similar test was run with Auto-Pickup disabled:

Full-Event Use Case 4 (Observing All Devices): Group Pickup, Auto-Pickup Disabled

Action	Events	Call info
provider. registerPickupAlert(N, P1) [Basic call happens, see Case 1] … pause for Pickup Group's delay Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C is dialing the PU number PU is removed from Call 2 C is removed from Call 2 Call 2 is destroyed	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP

Action	Events	Call info
C gets a connection on Call 1 B is dropped from Call 1 C is ringing C picks up	GC1-ConnCreatedEvent[ADDRS] GC1-ConnInprogressEvent GC1-CallCtlConnOfferedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent GC1-CallCtlConnDisconnectedEv GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Observing Only Device C: Auto-Pickup Enabled

Actions	Call events	Call IDs/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes pickup(Address of C) at Terminal for C C is connected to Call 1 C is dropped from Call 2 Call 2 is invalidated / cleared A and C are connected on Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-CallCtlConnEstablishedEv-C	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REAON_CALLPICKUP CCalling A, CCalled: NONE LRP: NONE CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_CALLPICKUP REASON_CALLPICKUP CCalling A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

Observing Only Device C: Auto-Pickup Disabled

Actions	Events	Call events\ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes pickup(Address of C) at Terminal for C Call 2 is destroyed C is added to Call 1, but does not pick up	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_NORMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP
C is ringing	GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	REASON_NORMAL
C picks up, and is connected to Call 1	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_NORMAL

Observing Only Device C: Group Pickup, Auto-Pickup Enabled

Actions	Call event	Call ID/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C dials the Pickup Number C is added to Call 1 PU is added to Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP CCalling: A, C Called: C CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP

Actions	Call event	Call ID/ Call info
PU # is removed from Call 2 C is removed from Call 2 Call 2 I invalidated / cleared C is connected to Call 1 PU is removed from Call 1	GC2-ConnDisconnectedEvent-PUv GC2-CallCtlTermConnDroppedEvvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEnde GC2-ConnDisconnectedE GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEdEv GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-PU GC1-CallCtlConnEstablishedEv-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU	CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL CCalling: A, CCalled: C REASON_CALLPICKUP CCalling: A, CCalled: C REASON_CALLPICKUP

Observing Only Device C: Group Pickup, Auto-Pickup Disable

Actions	Call event	Call ID/ Call info
provider. registerPickupAlert(N, P1) Provider receives Pickup Alert Application invokes directedPickup(Address of C, N) at Terminal for C C dials the PU Number PU is dropped from Call 2 C is dropped from from Call 2 Call 2 is destroyed C is added to Call 1	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC1-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A	Calling A, Called B, PUG Number N, PUG partition P1 CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP REASON_CALLPICKUP REASON_NOTMAL CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP

Actions	Call event	Call ID/ Call info
C is ringing C is connected to A	GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	REASON_NORMAL

Invoking Pickup on a Ringing Shared Line (CSCsy30964)

This is an odd scenario that normal applications will probably not see.

A calls a shared line DN (B), that has SLT1 and SLT2 on it. B is in pickup group N, P1.

B and B’ ring, and instead of invoking answer(), the application invokes pickup() on B’.

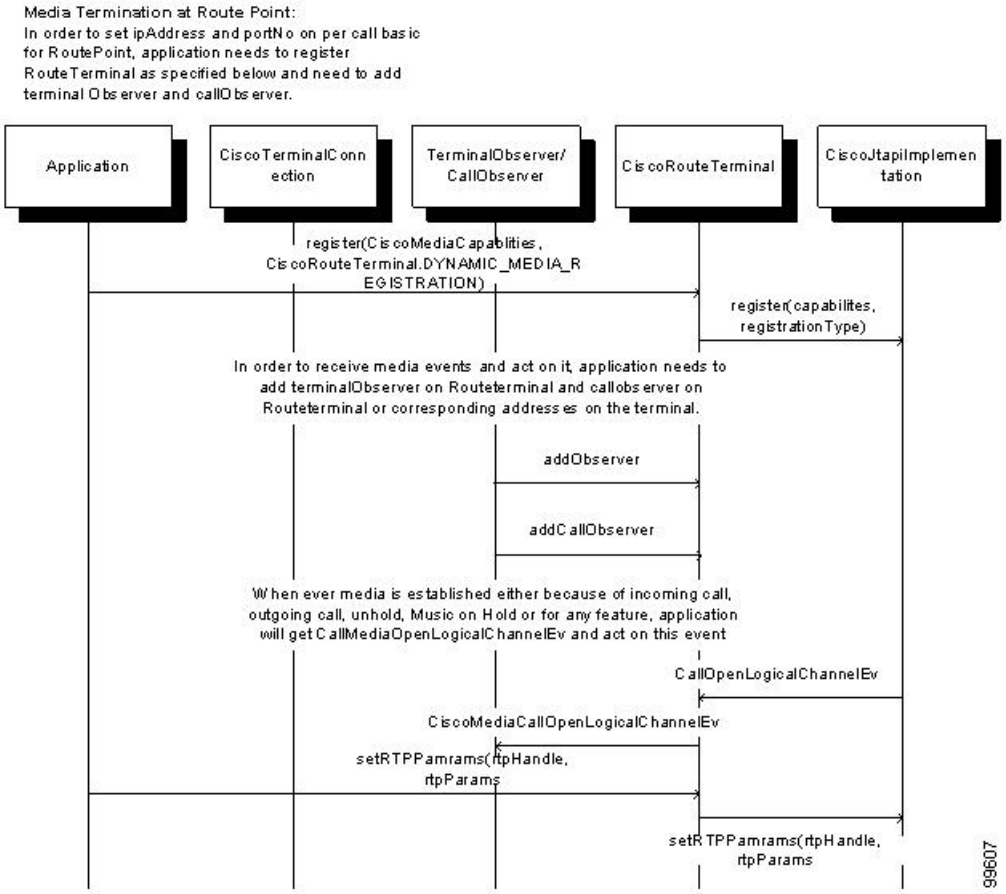
Action	Call event	Call ID/Info
provider. registerPickupAlert(N, P1) A goes offhook and dials B (Basic Call) B is ringing on both shared lines	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnInprogressEvent GC1-ConnAlertingEvent GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1- CallCtlTermConnRingingEv	CCalling: A, CCalled: NONE Calling: A, Called: NONE CAUSE_NEW_CALL REASON_NORMAL LRP: NONE CCalling: A, CCalled: B Calling: A, Called: B

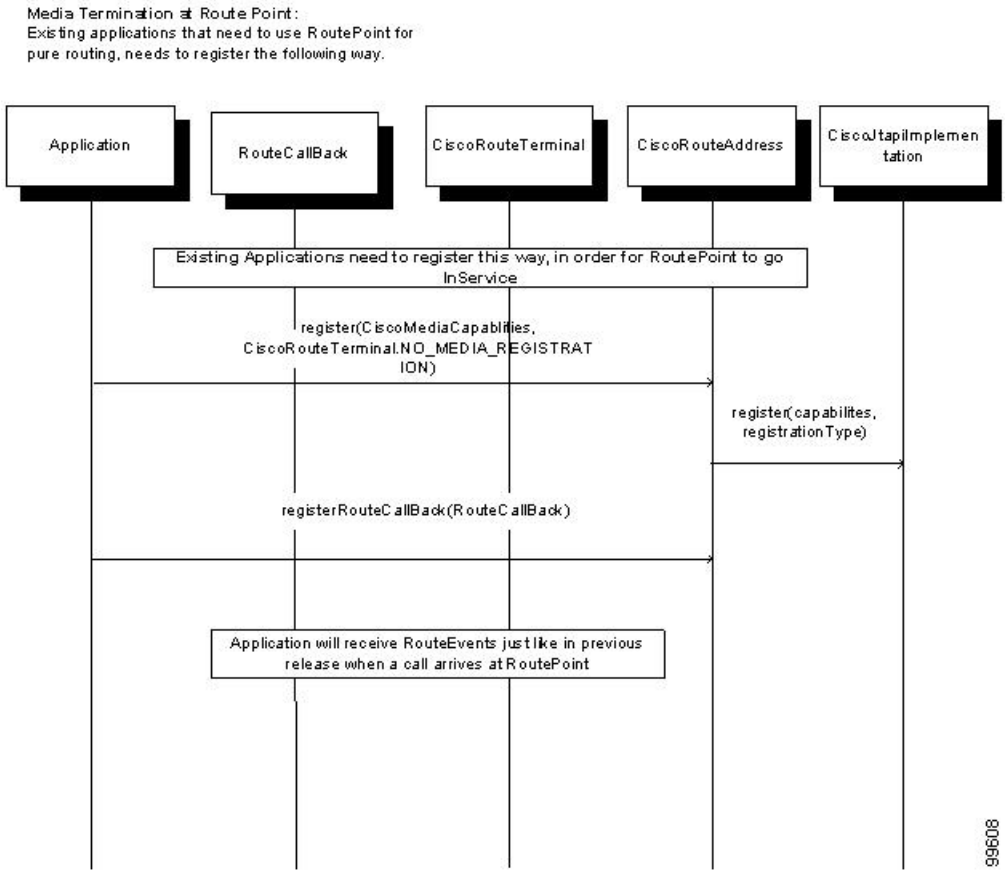
Action	Call event	Call ID/Info
… pause for Pickup Group's delay Provider receives Pickup Alert Application invokes Terminal.pickup(Address of B) on Terminal of SLT2 Shared line gets a connection on GC1 GC2 gets cleaned up SLT1 on Call 1 goes passive/bridged SLT2 on Call2 goes active SLT2 is talking	CiscoProvPickupCallAlertEvent GC2-CallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnCreatedEvent GC2- TermConnPassiveEvent GC2-CallCtlTermConnInUseEv GC2-CiscoCallChangedEv GC1-ConnConnectedEvent GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-CiscoCallChangedEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent GC2-CallCtlConnDisconnectedEv GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnPassiveEvent GC1-CallCtlTermConnBridgedEv GC1-CallCtlConnEstablishedEv GC1-CallCtlTermConnInUseEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	Calling A, Called B, PUG Number N, PUG partition P1 CCalling C, CCalled: NONE LRP: NONE REASON_NORMAL CCalling A, CCalled: B Calling: A, Called: B, LRP: B REASON_CALLPICKUP

Media Termination at Route Point

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

Message Sequence Charts





Mobility Interaction Support

Pre-conditions to mobility interaction use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Enable "Route calls to all remote destinations" checkbox for CTIRD1 and CTIRD3.
- CTIRD1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "C1_CTIRD1_RDD3", Number: "339007")
- CTIRD3 associated to user "Mobility3", dn = 9202
    - Remote destination 1 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- RDP1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "C1_CTIRD1_RDP1", Number: "334007")
- RDP3 associated to user "Mobility3", dn = 9202
    - Remote destination 1 (Name: "C1_CTIRD3_RDP1", Number: "334003")
    - Remote destination 2 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- Device A (IP Phone - Name: "SEP2401C7824EA3", Line A1 (dn: 9000))

- User1 has in its control list: Device A, CTIRD1 and CTIRD3. All devices and lines are observed.

Table 16: Call to CTI RD When App Not Active with Unique RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
A1 calls CTIRD1 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303"). Call is answered at the RD of RDP1 (dn = 4007).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 *Call is offered to the remote destination of RDP1 after delay. Call is not offered to CTIRD1 or to the remote destination of CTIRD1 (dn = 9007).	

Table 17: Call to CTI RD When App Active with Unique RD with Answer on RD of RDP

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD1 to be 339007. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339007", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339007" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1	
Call is answered at the RD of RDP1 (dn = 4007).	GC1: CallCtlTermConnRingingEv CTIRD1 *Call is offered to CTIRD1 and to the RD of CTIRD1 without delay. Call is offered to RD of RDP1 after delay. GC1: TermConnDroppedEv CTIRD1 GC1: CallCtlTermConnDroppedEv CTIRD1	

Table 18: Call to CTI RD When App Active with Unique RD with Answer on RD of CTI RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD1 to be 339007. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339007", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339007" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "2303").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 *Call is offered to CTIRD1 and to the RD of CTIRD1 without delay. Call is offered to RD of RDP1 after delay.	
Call is answered at the RD of CTIRD1 (dn = 9007).	GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1	

Table 19: Call to CTI RD When App Not Active with Shared and Unique RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
A1 calls CTIRD3 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202"). Call is answered at the RD of RDP3 (dn = 4003).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 *Call is offered to the remote destination of RDP3 after delay. Since 339006 is shared between CTIRD3 and RDP3, call is only seen at 339006 after delay.	

Table 20: Call to CTI RD When App Active with Shared and Unique RD with Answer on RD of CTI RD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
App sets active rd of CTIRD3 to be 339006. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339006", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339006" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.

Action	Events	Call Info
A1 calls CTIRD3 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 *Call is offered to CTIRD3 and to the RD of CTIRD3 (which is shared with RDP3) without delay. Call is offered to the unique RD of RDP3 after delay.	
Call is answered at the RD of CTIRD3 (dn = 9006). This is the RD that is shared between CTIRD3 and RDP3.	GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	

Table 21: Call to CTI RD When App Active with Shared and Unique RD with Answer on RD of RDP

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
App sets active rd of CTIRD3 to be 339006. User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("339006", true).	CiscoProvTerminalRemoteDestinationChangedEv	CiscoRemoteTerminal. getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "339006" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
A1 calls CTIRD3 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 *Call is offered to CTIRD3 and to the RD of CTIRD3 (which is shared with RDP3) without delay. Call is offered to the unique RD of RDP3 after delay.	
Call is answered at the RD of RDP3 (dn = 4003).	GC1: TermConnDroppedEv CTIRD3 GC1: CallCtlTermConnDroppedEv CTIRD3	

Modifying Calling Number

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

Scenario One

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

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

A calls RP.

Call arrives at RP

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

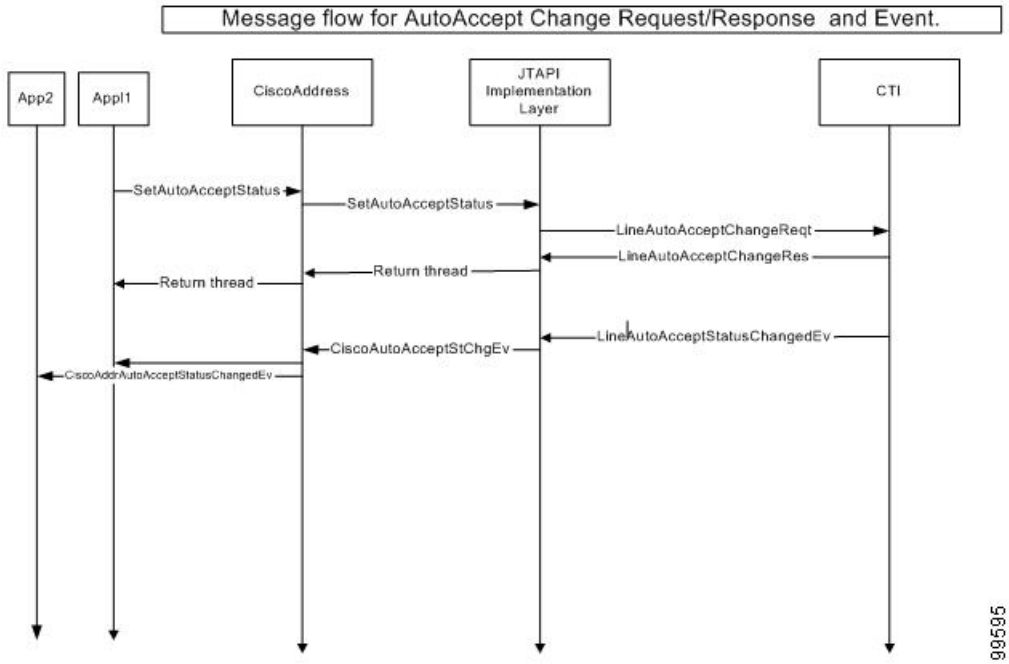
Scenario Two

The application is controls A and B.

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

Action	Event	Fields
A calls RP, which is not in controlled list.	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CallCtlConnEstablishedEv A ConnCreatedEv RP ConnInProgressEv RP CallCtlConnOfferedEv RP	getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = getModifiedCallingAddress() = A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = getModifiedCallingAddress() = A getModifiedCalledAddress() = getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = A getModifiedCalledAddress() = B
Another application controls the RP selectRoute to B with modifying calling number as M.	NEW META EVENT_____ META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnDisconnectedEv RP CallCtlConnDisconnectedEv RP NEW META EVENT_____ META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B
B answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress () = RP getCurrentCallingAddress () = A getCurrentCalledAddress() = B getModifiedCallingAddress() = **M** getModifiedCalledAddress() = B

AutoAccept for CTIPort and RoutePoint



Silent Monitoring Use Cases

A and TA are address and terminal of monitor target or recording initiator

B and TB are address and terminal of monitor initiator.

Scenario One

Administrator enables monitoring capability for the user.

Action	Events	Call info
	CiscoProviderCapabilityChangedEv hasMonitorCapabilityChanged() on this event returns true hasRecordingCapabilityChanged() returns true	NA
ciscoProvider.getCapabilities().canMonitor()	JTAPI returns true	NA

Scenario Two

Start and Stop monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. Application has monitoring capability enabled.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A or GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, terminal name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorInitiatorInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A GC2: Calling: B Called: A LRP: null Current calling: B Current called: A

Action	Events	Call info
A puts the call on hold A resumes the call B calls drop on GC2 to stop monitoring	GC2: CiscoRTPOutputStoppedEv GC1: CiscoRTPOutputStoppedEv **GC1: CallCtlTermConnHeldEv** TA **GC2**:CiscoRTPInputStoppedEv **GC1**: CiscoRTPInputStoppedEv **GC1**: CiscoRTPOutputStartedEv GC2: CiscoRTPOutputStartedEv GC1: CallCtlTermConnTalking TA **GC2**:CiscoRTPInputStartedEv **GC1:** CiscoRTPInputStartedEv GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnEv A GC1: CiscoTermConnMonitorEndEv TA GC2: ConnDisconnEv B GC2: CallInvalidEv	

Scenario Three

Monitor initiator transfers the call to Y. A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. application has monitoring capability enabled. B transfers the call to Y.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 and ci2 giving CI1, A and TermA from GC1 Or using the teminalconnection of A.	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: ConnConnectedE**v for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A or GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL Monitor_TARGET address:A, device name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv TA** **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_NORMAL address:B, device name: TB rtphandle = ci2	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Action	Events	Call info
B consults Y using GC3 and completes transfer. Call observer on Y would see	GC3**:** CallActiveEv GC3: ConnConnectedEv B GC3: CallCtlTermConnTalkingEv TB GC3: ConnConnectedEv Y CiscoTransferStartEv(GC3->GC2) GC3: ConnDisconnectedEv Y GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:Y, device name: TY GC3: ConnDisconnectedEv B GC3: CallCtlTermConnDroppedEv TB GC3: CallInvalidEv GC2: CallCtlTermConnDroppedEv TB …. GC2: CallInvalidEv CiscoTransferEndEv GC3: CallActive **GC3:** CallCtlTermConnRingingEv TY GC3: CallCtlTermConnTalkingEv TY CiscoTransferStartEv CiscoCallChangedEv GC3->GC2 GC2: ConnConnectedEv Y **GC2**: ConnConnectedEv B GC2: CiscoTermConnMonitorTargetInfoEv TY Cause: CAUSE_NORMAL address:A, device name: TA GC3: ConnDisconnectedEv Y **GC3:** CallCtlTermConnDroppedEv TY GC3: CallInvalidEV GC2: ConnConnectedEv X GC2: ConnDisconnectedEv A CiscoTransferEndEv (**CiscoTermConnMonitorInitiatorInfoEv** on GC1 is independent of transfer events on GC3 and GC2 and can be delivered at any time before or after end event.)	GC1: Calling: X Called: A LRP: A Current calling: X Current called: Y

Scenario Four

Monitoring a barged call: A and A' are shared lines. Caller calls A, A answers the call. A' barge into the call. B calls start monitor.

Action	Events	Call info
A answers GC1 A' barges the call B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv GC1: CallCtlTermConnBridgedEv TermA' GC1: **GC1:CallCrlTermConnTalkingEv** TA' Exception is thrown to startMonitor request.	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Five

Monitor and recording: A is monitor target and has auto recording configured. B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on both A and B. Application has monitoring capability enabled.

Action	Events	Call info
A answers GC1 B calls start monitor using GC2 giving CI1, A and TermA from GC1	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv GC1: CiscoTermConnRecordingStartEv TA **GC1: CiscoTermConnRecordingTargetInfoEv TA** CiscoRTPInputStartedEv **GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A (No terminal connection for A on GC2) GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, device name: TA, rtphandle = CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorTargetInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Six

Observing remote in use shared line in Monitoring: A and A' are shared lines. Caller calls A, A answers the call. B calls start monitor. Application has call observer on A' only. B initiates monitor request for connected call on A. No start events are delivered to call observer of A'.

Action	Events	Call info
A answers GC1 and B initiates monitor A puts the call on HOLD A' answers the call B drops the call and initiates start monitor using GC3 giving the terminal connection of A' in monitor request.	**CallActiveEv** for callID = GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A' Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A' Cause: CAUSE_NORMAL GC1: ConnConnectedEv X … **GC1:CallCrlTermConnRingingEv** TA' Cause: CAUSE_NORMAL GC1: CallCtlTermConnBridgedEv TermA' Cause: CAUSE_NORMAL **GC1: CallCrlTermConnHeldEv TA**' **GC1:CallCrlTermConnTalkingEv TA**' GC1: CiscoTermConnMonitorStartEv TA' **GC1: CiscoTermConnMonitorInitiatorInfoEv TA'** Cause: CAUSE_NORMAL address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Scenario Seven

Snap Shot events for Monitor and recording: A is monitor target and has auto recording configured. B is monitor initiator. X calls A, A answers the call GC1 (ci1), B calls start monitor using GC2. Another application adds call observer on A after monitoring and recording sessions are established.

Action	Events	Call info
	CallActiveEv for callID = GC1 Cause: CAUSE_SNAPSHOT **GC1:ConnCreatedEv** for A Cause: CAUSE_SNAPSHOT **GC1:ConnConnectedEv** for A Cause: CAUSE_SNAPSHOT GC1: ConnConnectedEv X … **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingTargetInfoEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_SNAPSHOT address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

Secured Monitoring Use Cases

Monitoring Use Cases

Scenario	Expected Result	Info
Scenario 1 1. Supervisor and agent's device security mode is encrypted 2. Customer and Agent are in a secured call. 3. Supervisor initiates a monitoring request.	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 TargetAddress = A TargetTerminal = TermA targetCall = GC1
4. The Agent Customer call is dropped.	GC1: CiscoTermConnMonitoringEndEv TermA … GC1: CallInvalidEv … GC2: CallInvalidEv	

Scenario	Expected Result	Info
Scenario 2 1. Supervisor and agent's device security mode is encrypted 2. Customer and Agent are in a non-secured call. 3. Supervisor initiates a monitoring request 4. Supervisor stops monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1

Scenario	Expected Result	Info
Scenario 3 1. Agent's Device is non-secured and supervisor is secured 2. Customer and Agent are in a non secured call 3. Supervisor initiates a monitoring request 4. Supervisor stops Monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
Scenario 4 1. Agent's device is secured and supervisor is non-secured 2. Customer and agent are in a secured/non-secured call 3. Supervisor initiates a monitoring request	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S .. GC2: CallCtlConnEstablished TermS GC2: ConnFailedEv S2 GC2: CallCtlConnFailedEv S2 PriviledgeViolationException :CTIERR_SECURITY_CAPABILITY_MISMATCH (as supervisor's does not meet the security capabilities of the Agent)	Cause: BCNAUTHORIZED

Scenario	Expected Result	Info
Scenario 5 1. Both supervisor and agent are non-secured 2. Customer and agent are in a non-secured call 3. Supervisor initiates a monitoring request 4. Supervisor stops monitoring	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: connCreatedEv A .. GC2: CallCtlConnEstablishedEv termA GC2: CiscoTermConnMonitoringTargetInfoEv GC1: CiscoTermConnMonitoringEndEv TermA … GC2: CallInvalidEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
Scenario 6 1. Supervisor1 and agent are secured, supervisor2 is non-secured. 2. Customer and Agent are in a secured/non-secured call 3. A secured monitoring session is in progress with supervisor1.	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitoringInitiatorInfoEv termA GC2 : ConnCreatedEv A .. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitoringTargetInfoEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1 cause : CAUSE_BCNAUTHORISED

Scenario	Expected Result	Info
4. Supervisor1 transfers the monitoring call to supervisor2	CiscoTransferStartEv ….. CiscoTransferEndEv The monitoring session is torn down (since supervisor2 does not meet the security capabilities of the agent) GC2: ConnFailedEv S2 GC2: CallCtlConnFailedEv S2 GC1: CiscoTermConnMonitorEndEv Events Received on Supervisor1 CiscoAddrMonitoringTerminatedEv	TransactionID : xxxx AgentAddr = A AgentDevice = termA AgentCID = agentCI in GC1 Supervisor1 Device Name = TermS1 Cause = BCNAuthorised
Scenario 7 1. Supervisor1 and agent are secured, supervisor2 is secured. 2. Customer and Agent are in a secured/non-secured call 3. A secured monitoring session is in progress with supervisor1.	GC1: CallCtlTermConnTalkingEv TermA GC2: ConnCreatedEv S ….. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv GC1: CiscoTermConnMonitoringInitiatorInfoEv termA GC2 : ConnCreatedEv A .. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitoringTargetInfoEv	Initiatortaddress = S InitiatorTerminal = TermS InitiatorCall = GC2 Targetaddress = A TargetTerminal = TermATarget call = GC1
4. Supervisor1 transfers the monitoring call to supervisor2	CiscoTransferStarteEv …. CiscoTransferEndEv GC2: ConnCreatedEv S2 ….. GC2: CallCtlConnEstablishedEv TermS2 GC2: CiscoTermConnMonitoringTargetInfoEv	Targetaddress = A TargetTerminal = TermATarget call = GC1

Monitoring and Recording Use Cases

Scenario	Expected Result
Scenario 1 1. Agent is non-secured, Supervisor and recorder are encrypted 2. Agent is in a non-secured call with customer 3. Supervisor monitors the call in non-secured mode	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS
4. Application requests for recording the call at Supervisor	CiscoTermConnRecordingStartEv CallCtlTermConnTalkingEv TermS The recording is done in a secured mode as Both the supervisor and recorder are secured, but the overall call security is non-secured.

Native Queuing

1. Hunt Pilot HP has one member B.
2. Check box for “Queue Calls” is checked.
3. “Maximum In Queue timer” is set as 60 seconds.
4. “Destination When Maximum Wait Time is Met” is set as C
5. Queue depth is set to 1
6. “Destination When Queue is Full” is set to C

Queuing of Call

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 ConnInProgessEv HP GC2 CallCtlConnQueuedEv HP GC2 CallCtlConnDisconenctedEv HP GC2 ConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.INTERNAL Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT

De-queuing of a call

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 ConnInProgessEv HP GC2 CallCtlConnQueuedEv HP GC2 CallCtlConnDisconenctedEv HP GC2 ConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.INTERNAL Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT

Action	Event	Call info
B disconnects the call	GC1 ConnDisconnectedEv HP GC1 CallCtlConnDisconnectedEv HP GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconenctedEv B GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconenctedEv A GC1 CallInvalidEv GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertinEv B GC2 CallCtlConnAleringEv B GC2 TermConnCreatedEv Term B Gc2 termConnRingingEv TermB GC2 CallCtlTermConnRingingEv TermB GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Connected.getAddress().getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = B Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
B Answers	GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnActiveEv Term B GC2 TermConnCtalkingEv TermB	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = B Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP

Maximum In-Queue Timer Expires

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Action	Event	Call info
After 60 seconds	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 CallCtltermConnRingingEv termC GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = C Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = C Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_TIMER_EXPIRED Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Maximum In-Queue Timer Expires with Destination as Another HP Whose Member E Is Free

All members of HP are busy. The destination when queue timer expires is HP2 whose members E is free

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
After 60s call gets offered on HP2	GC2 ConnCreatedEv E GC2 ConnInProgressEv E GC2 CallCtlConnOfferedEv E GC2 CiscoHuntConnCreatedEv HP2 GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E Gc2 termConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE GC2 ConnConnectedEv HP2 GC2 CallCtlConnEstablishedEv HP2 GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING_TIMER_EXPIRED Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = B Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING_TIMER_EXPIRED (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
E answers	GC2 ConnConnectedEv E GC2 CallCtlConnEstablishedEv E GC2 TermConnActiveEv Term E GC2 TermConnCtalkingEv TermE	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP

Maximum In-Queue Timer Expires with Destination as Another HP Whose Members Are Busy

All members of HP are busy. The destination when queue timer expires is HP2 whose members (E) is also busy

Action	Event	Call info
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()). getAddress()). getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
E becomes Free after >60s	GC2 ConnCreatedEv E GC2 ConnInProgressEv E GC2 CallCtlConnOfferedEv E GC2 CiscoHuntConnCreatedEv HP2 GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E Gc2 termConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE GC2 ConnConnectedEv HP2 GC2 CallCtlConnEstablishedEv HP2 GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING Ev.getConnection()). getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()). getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING (AddressImpl)(ConnectionImpl) ((ConnEvImpl) Ev.getConnection()). getAddress()).getType() = CiscoAddress.INTERNAL
E answers	GC2 ConnConnectedEv E GC2 CallCtlConnEstablishedEv E GC2 TermConnActiveEv Term E GC2 TermConnCtalkingEv TermE	Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = HP

Queue Is Full

Action	Event	Fields
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Fields
E calls HP Call gets offered on C	GC3 CallActiveEv GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 CallCtlConnEstablishedEv E GC3 TermConnCreatedEv E GC3 CallCtlTermConnTalkingEv TermE GC3 CiscoHuntConnCreatedEv HP GC3 ConnCreatedEv C GC3 CallCtlCallOfferedEv C GC3 ConnAlertinEv C GC3 CallCtlConnAleringEv C GC3 TermConnCreatedEv Term C Gc3 termConnRingingEv TermC GC3 CallCtlTermConnRingingEv TermC GC3 CallCtlConnDisconnectedEv HP GC3 ConnCisconnectedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_BUSY Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_BUSY
C answers	GC3 CallCtlConnEstablishedEv C GC3 CallCtlTermConnTalkingEv termC	Call.getCalledAddress() = HP Call.getCallingAddress() = E Call.getCurrentCalledAddress() = C Call.getCurrentCallingAddress() = E Call.getModifiedCallingAddress() = E Call.getModifiedCalledAddress() = C Call.getLastRedirectedAddress() = HP

When Disconnect Is Selected for Queue Full

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC1 ConnConenctedEv HP GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 CallCtltermConnRingingEv termB GC1 CallCtlConnEstablishedEv HP	
B answers the call	GC1 CallCtlConnEstablishedEv B GC1 CallCtlTermConnTalkingEv termB	
D calls HP (Call gets Queued)	GC2 CallActiveEv GC2 ConnCreatedEv D GC2 CallCtlConnInitiatedEv D GC2 CallCtlConnEstablishedEv D GC2 TermConnCreatedEv D GC2 CallCtlTermConnTalkingEv TermD GC2 CiscoHuntConnCreatedEv HP GC2 ConnCreatedEv HP GC2 CallCtlCallOfferedEv HP GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_QUEUING Call.getCalledAddress() = HP Call.getCallingAddress() = D Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = D Call.getModifiedCallingAddress() = D Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
E calls HP Call fails	GC3 CallActiveEv GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 ConnCreatedEv E GC3 CallCtlConnInitiatedEv E GC3 CallCtlConnEstablishedEv E GC3 TermConnCreatedEv E GC3 CallCtlTermConnTalkingEv TermE GC3 CiscoHuntConnCreatedEv HP GC3 ConnFaileEv E GC3 CallCtlConnFailedEv E Gc3 ConnDisconenctedEv HP Gc3 ConnDisconnectedEv E Gc3 CallCtlConnDisconenctedEv E Gc3 CallInvalidEv	Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_NORMAL Cause = UserBusy

Same result as above is observed when "Disconnect" is selected for MaxQueueTime and Agents Not Logged in configs and those conditions get hit.

No Agents Are Logged In

Action	Event	Call info
A calls HP	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 CallCtlConnInitiatedEv A GC1 CiscoHuntConnCreatedEv HP GC1 CallCtlConnEstablishedEv A GC1 TermConnCreatedEv A GC1 CallCtlTermConnTalkingEv TermA GC3 ConnCreatedEv C GC3 CallCtlCallOfferedEv C GC3 ConnAlertinEv C GC3 CallCtlConnAleringEv C GC3 TermConnCreatedEv Term C Gc3 termConnRingingEv TermC GC3 CallCtlTermConnRingingEv TermC GC3 CallCtlConnDisconnectedEv HP GC3 ConnCisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_UNAVAILABLE Ev.getCiscoFeatureReason() = CiscoFeatureReason. REASON_DEQUEUING_AGENTS_UNAVAILABLE Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT
C answers the call	GC1 CallCtlConnEstablishedEv C GC1 CallCtlTermConnTalkingEv termC	

Caller Redirects While in Queue

A calls HP, call offered on B and B answers (GC1). D calls HP and gets queued (GC2).

Action	Event	Call info
D redirects the call to E	GC2 ConnCreatedEv E GC2 CallCtlCallOfferedEv E GC2 TermConnDroppedEv TermD GC2 CallCtlTermConnDroppedEv termD GC2 ConnDisconnectedEv D GC2 CallCtlTermConnDisconnectedEv D GC2 ConnAlertinEv E GC2 CallCtlConnAleringEv E GC2 TermConnCreatedEv Term E GC2 TermConnRingingEv TermE GC2 CallCtlTermConnRingingEv TermE	Ev.getCiscoFeatureReason() = REASON_REDIRECT Call.getCalledAddress() = E Call.getCallingAddress() = HP Call.getCurrentCalledAddress() = E Call.getCurrentCallingAddress() = HP Call.getModifiedCallingAddress() = HP Call.getModifiedCalledAddress() = E Call.getLastRedirectedAddress() = D
E answers	GC2 CallCtlConnEstablishedEv E GC2 CallCtlTermConnTalkingEv termE	
B drops GC1	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlTermConnDisconnectedEv A GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlTermConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertinEv B GC2 CallCtlConnAleringEv B GC2 TermConnCreatedEv Term B Gc2 termConnRingingEv TermB GC2 CallCtlTermConnRingingEv TermB GC2 ConnConnectedEv HP GC2 CallCtlConnEstablishedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Call.getCalledAddress() = HP Call.getCallingAddress() = E Call.getCurrentCalledAddress() = HP Call.getCurrentCallingAddress() = E Call.getModifiedCallingAddress() = E Call.getModifiedCalledAddress() = HP Call.getLastRedirectedAddress() = HP

Action	Event	Call info
	GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP	Ev.getCiscoFeatureReason() = CiscoFeatureReason.REASON_DEQUEUING (AddressImpl)(ConnectionImpl)((ConnEvImpl) Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL
B answers	GC2 CallCtlConnEstablishedEv B GC2 CallCtlTermConnTalkingEv termB	

Caller (Observed) Conferences While in Queue

A calls HP, call offered on B and B answers (GC1). D calls HP and gets queued (GC2).

Action	Event	Call info
D initiates a consult call with E	GC2 callCtltermConnHeldEv termD GC3 CallActiveEv GC3 ConnCreatedEv D GC3 CallCtlConnInitiatedEv D GC3 CallCtlConnEstablishedEv D GC3 TermConnCreatedEv D GC3 CallCtlTermConnTalkingEv TermD GC3 ConnCreatedEv E GC3 ConnInProgressEv E GC3 CallCtlConnOfferedEv E GC3 ConnAlertingEv E GC3 CallCtlConnAlertingEv E GC3 CallCtltermConnRingingEv termE	
E answers	GC2 CallCtlConnEstablishedEv E GC2 CallCtlTermConnTalkingEv termE	

Action	Event	Call info
D conferences GC1 with Cg2 GC1.conference(GC2)	GC2 CiscoConfereceStartedEv GC3 TermConnDroppedEv TermE GC3 CallCtlTermConnDroppedEv termE GC3 ConnDisconnectedEv E GC3 CallCtlTermConnDisconnectedEv E GC3 TermConnDroppedEv TermD GC3 CallCtlTermConnDroppedEv termD GC3 ConnDisconnectedEv D GC3 CallCtlTermConnDisconnectedEv D GC3 CallInvalidEv GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEv HP GC2 ConnCreatedEv HP GC2 CallCtlConnEstablishedEv HP GC2 CiscoConferenceEndEv	Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getConnection()).getAddress()).getType() = CiscoAddress.UNKNOWN

Action	Event	Call info
B drops GC1	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv termA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv termB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEvHP GC2 ConnCreatedEv B GC2 ConnInProgressEv B GC2 CallCtlConnOfferedEv B GC2 CiscoHuntConnCreatedEv HP GC2 ConnAlertingEv B GC2 CallCtlConnAlertingEv B GC2 CallCtltermConnRingingEv termB Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP GC2 ConnCreatedEv HP Gc2 ConnAlertingEv HP GC2 CallCtlConnAlertingEv HP GC2 ConnDisconnectedEv HP GC2 CallCtlConnDisconnectedEvHP	Ev.getCiscoFeatureReason() = REASON_NORMAL Ev.getConnection()).getAddress()).getType() = CiscoAddress.UNKNOWN Ev.getCiscoFeatureReason() = REASON_DEQUEUING Ev.getCiscoFeatureReason() = REASON_DEQUEUING Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL Ev.getCiscoFeatureReason() = REASON_CONFERENCE Ev.getConnection()).getAddress()).getType() = CiscoAddress.HUNT_PILOT
B answers	Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP Gc2 ConnConenctedEv HP GC2 CallCtlConnEstablished HP GC2 TermConnActiveEv termb GC2 CallCtlTermConnTalkingEv termB	Ev.getConnection()).getAddress()).getType() = CiscoAddress.INTERNAL

Use Cases for NuRD (Number Matching for Remote Destination)

Prerequisites

Pre-conditions to nurd use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Enable "Route calls to all remote destinations" checkbox for CTIRD1 and CTIRD2.
- Clusterwide service parameter "Reroute Remote Destination Calls to Enterprise Number" is set to true.
- ICT Trunk configured with Route pattern is 408XXXXX with no discard digits.
- SIP Trunk configured with Route pattern is 409XXXXX with no discard digits.
- ICT Trunk configured with Route pattern is 33.XXXX with discard digits pre-dot.
- CTIRD1 associated to user "Mobility1", dn = 2303
    - Remote destination 1 (Name: "RDD1", Number: "40822077")
    - Remote destination 2 (Name: "RDD2", Number: "40922078")
- CTIRD2 associated to user "Mobility2", dn = 9200
    - Remote destination 1 (Name: "RDD3", Number: "40812115")
    - Remote destination 2 (Name: "RDD4", Number: "40912116")
- CTIRD2 has a shared ip phone D
- RDP2 associated to user "Mobility2", dn = 9200 with display and Unicode name configured to be "RDP2".
    - Remote destination 1 (Name: "RDD3", Number: "40812115"). This is shared with CTIRD2.
    - Remote destination 2 (Name: "RDDP1", Number: "40922095")
- Device A (IP Phone - Name: "SEP2401C7824EA3", Line A1 (dn: 9000)).
- Device B (IP Phone - Name: "SEP2401C7824EAE", Line B1 (dn: 9001)).
- User1 has in its control list: Device A, B, CTIRD1 and CTIRD2. All devices and lines are observed.

Basic Calls Initiated From Remote Destination

Table 22: Remote Destination Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDD1 calls A1 in which active rd is not set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConOfferedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Direct call between RDD1 and A.	CallingAddress = 40822077, CalledAddress = 9000, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = External

Table 23: Remote Destination Initiates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1 in which active rd is set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1 so looks like CTIRD1 is initiating a call to A.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = Internal

Table 24: Remote Destination Initiates Call with Active RDD with Only CTIRD Observed

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 observes only CTIRD		
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.	*If active remote destination is not set, then app will not see any call events as it would be direct call between RDD1 and A1.	

Action	Events	Call Info
RDD1 calls A1 in which active rd is set. Call is answered at A1 (dn = 9000).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 *Call is routed through CTIRD1 so looks like CTIRD1 is initiating a call to A.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000 CurrentCallingAddress.getType () = Internal

Basic Calls to Remote Destination

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
A1 calls RDD1 in which active rd is not set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnNetworkReachedEv 40822077 GC1: CallCtlConnNetworkAlertingEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 *Direct call between A and RDD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 40822077 CurrentCalledAddress.getType() = External

Table 25: Call to Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through and offered on CTIRD1 and extended to RDD1 with no delay.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303 CurrentCalledAddress.getType() = Internal

Table 26: Call to Remote Destination with Active RDD with Only CTIRD Observed

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 observes only CTIRD		
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("40822077", true) on CTIRD1.	*If active remote destination is not set, then app will not see any call events as it would be direct call between A1 and RDD1.	

Action	Events	Call Info
A1 calls RDD1 in which active rd is set. User1 invokes Call.connect("SEP2401C7824EA3", "9000", "40822077"). Call is answered at RDD1 (dn = 40822077).	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through and offered on CTIRD1 and extended to RDD1 with no delay.	CallingAddress = 9000, CalledAddress = 2303, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303 CurrentCalledAddress.getType() = Internal

CTIRD/RDP Interaction

Table 27: Remote Destination Shared Between CTIRD and RDP Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds	ProvInServiceEv	

Action	Events	Call Info
RDD3 calls B1 in which active rd is not set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnInitiatedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through the RDP.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName = RDP2

Table 28: Remote Destination Shared Between CTIRD and RDP Initiates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD3 (dn = 40812115). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40812115", true) on CTIRD2.		

Action	Events	Call Info
RDD3 calls B1 in which active rd is set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: CallCtlConnDialingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through CTIRD2 so looks like CTIRD2 is initiating a call to B.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 29: Remote Destination Unique to CTIRD2 Initates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDD4 calls B1 in which active rd is not set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnCreatedEv 40912116 GC1: ConConnectedEv 40912116 GC1: CallCtlConnEstablishedEv 40912116 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TemConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlTermConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD4 and B1.	CallingAddress = 40912116, CalledAddress = 9001, CurrentCallingAddress = 40912116, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 30: Remote Destination Unique to CTIRD2 Initates Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD4 (dn = 40912116). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40912116", true) on CTIRD2.		

Action	Events	Call Info
RDD4 calls B1 in which active rd is set. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: CallCtlConnDialingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through CTIRD2 so looks like CTIRD2 is initiating a call to B.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName =

Table 31: Remote Destination Unique to RDP Initiates Call with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
RDDP1 calls B1. Call is answered at B1 (dn = 9001).	GC1: CallActiveEv GC1: ConnCreatedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnInitiatedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 GC1: CallCtlConnEstablishedEv 9200 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE *Call is routed through the RDP.	CallingAddress = 9200, CalledAddress = 9001, CurrentCallingAddress = 9200, CurrentCalledAddress = 9001 CurrentCallingPartyDisplayName = RDP2

Table 32: Call to Remote Destination Shared with RDP with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDD3 in which active rd is not set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40812115"). Call is offered to RDD3 after delay and then answered at RDD3 (dn = 40812115).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 *Call is routed through RDP.	CallingAddress = 9001, CalledAddress = 40812115, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName = RDP2

Table 33: Call to Remote Destination Shared with RDP with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD3 (dn = 40812115). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40812115", true) on CTIRD2.		

Action	Events	Call Info
B1 calls RDD3 in which active rd is set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40812115"). Call is answered at RDD3 (dn = 40812115).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnAlertingEv 9200 GC1: CallCtlConnAlertingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnRingingEv CTIRD2 GC1: CallCtlTermConnRingingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 *Call is routed through and offered on CTIRD2 and extended to RDD3 with no delay.	CallingAddress = 9001, CalledAddress = 40812115, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName =

Table 34: Call to Remote Destination Unique to RDP with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDDP1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40922095"). Call is offered to RDDP1 after delay and then answered at RDDP1 (dn = 40922095).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnInUseEv SEP2401C7824EA3 *Call is routed through RDP.	CallingAddress = 9001, CalledAddress = 40922095, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName = RDP2

Table 35: Call to Remote Destination Unique to CTIRD with No Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
B1 calls RDD4 in which active rd is not set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40912116"). Call is answered at RDD4 (dn = 40912116).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 40912116 GC1: ConnConnectedEv 40912116 GC1: CallCtlConnNetworkReachedEv 40912116 GC1: CallCtlConnNetworkAlertingEv 40912116 GC1: CallCtlConnEstablishedEv 40912116 *Direct call between B1 and RDD4.	CallingAddress = 9001, CalledAddress = 40912116, CurrentCallingAddress = 9001, CurrentCalledAddress = 40912116 CurrentCalledPartyDisplayName =

Table 36: Call to Remote Destination Unique to CTIRD with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD2 to be RDD4 (dn = 40912116). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40912116", true) on CTIRD2.		

Action	Events	Call Info
B1 calls RDD4 in which active rd is set. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40912116"). Call is answered at RDD4 (dn = 40912116).	GC1: CallActiveEv GC1: ConnCreatedEv 9001 GC1: ConnConnectedEv 9001 GC1: CallCtlConnInitatedEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE GC1: CallCtlConnDialingEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: ConnCreatedEv 9200 GC1: ConnInProgressEv 9200 GC1: CallCtlConnOfferedEv 9200 GC1: ConnAlertingEv 9200 GC1: CallCtlConnAlertingEv 9200 GC1: TermConnCreatedEv CTIRD2 GC1: TermConnRingingEv CTIRD2 GC1: CallCtlTermConnRingingEv CTIRD2 GC1: ConnConnectedEv 9200 GC1: CallCtlConnEstablishedEv 9200 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnPassiveEv SEP2401C7824EA3 GC1: CallCtlTermConnBridgedEv SEP2401C7824EA3 GC1: TermConnActiveEv CTIRD2 GC1: CallCtlTermConnTalkingEv CTIRD2 *Call is routed through and offered on CTIRD2 and extended to RDD4 with no delay.	CallingAddress = 9001, CalledAddress = 40912116, CurrentCallingAddress = 9001, CurrentCalledAddress = 9200 CurrentCalledPartyDisplayName =

Multiple Calls

Table 37: Make Multiple Calls From Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000

Action	Events	Call Info
RDD1 calls B1. Call is answered at B1.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnInProgressEv 9001 GC2: CallCtlConnOfferedEv 9001 GC2: ConnCreatedEv 40822077 GC2: ConnConnectedEv 40822077 GC2: CallCtlConnEstablishedEv 40822077 GC2: ConnAlertingEv 9001 GC2: CallCtlConnAlertingEv 9001 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3 GC2: ConnConnectedEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD1 and B1.	CallingAddress = 40822077, CalledAddress = 9001, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9001

Table 38: Make Multiple Calls To Remote Destination with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303

Table 39: Remote Destination First Makes a Call and Then Receives a Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: CallCtlConnDialingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Call is routed through CTIRD1.	CallingAddress = 2303, CalledAddress = 9000, CurrentCallingAddress = 2303, CurrentCalledAddress = 9000

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303

Table 40: Remote Destination First Receives a Call and Then Makes a Call with Active RDD

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is routed through CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
RDD1 calls B1. Call is answered at B1.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnInProgressEv 9001 GC2: CallCtlConnOfferedEv 9001 GC2: ConnCreatedEv 40822077 GC2: ConnConnectedEv 40822077 GC2: CallCtlConnEstablishedEv 40822077 GC2: ConnAlertingEv 9001 GC2: CallCtlConnAlertingEv 9001 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3 GC2: ConnConnectedEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE *Direct call between RDD1 and B1.	CallingAddress = 40822077, CalledAddress = 9001, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9001

Table 41: Persistent Call Exists and Remote Destination Initiates Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
Create persistent call on CTIRD1.		

Action	Events	Call Info
RDD1 calls A1. Call is answered at A1.	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnInProgressEv 9000 GC1: CallCtlConnOfferedEv 9000 GC1: ConnCreatedEv 40822077 GC1: ConnConnectedEv 40822077 GC1: CallCtlConnEstablishedEv 40822077 GC1: ConnAlertingEv 9000 GC1: CallCtlConnAlertingEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnRingingEv SEP2401C7824EA3 GC1: CallCtlTermConnRingingEv SEP2401C7824EA3 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 *Direct call between RDD1 and A1	CallingAddress = 40822077, CalledAddress = 9000, CurrentCallingAddress = 40822077, CurrentCalledAddress = 9000
Disconnect the persistent call. Call disconnects successfully.		
A drops the call. Call disconnects successfully.		

Table 42: Persistent Call Exists and Make Call to Remote Destination

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
Create persistent call on CTIRD1.		

Action	Events	Call Info
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Call is offered on CTIRD1.	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303
Disconnect the persistent call. Disconnect throws exception.		Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_DISCONNECT_PERSISTENT_CALL_FAILED_CALL_ACTIVE
A drops the call. Call disconnects successfully.		
Disconnect the persistent call. Call disconnects successfully.		

Table 43: Call to Remote Destination with Active RDD and Call Forward All Configured on CTIRD with Only CTIRD Observed

Action	Events	Call Info
Configure CallForwardAll on CTIRD1 to 2302.		
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1 *Nurd features disables CFA.	CallingAddress = 9000, CalledAddress = 2303, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Table 44: Max Calls Limit Reached Where CTIRD Has 2 Calls to the Remote Destination and CTIRD Still Tries to Make a Call

Action	Events	Call Info
Set Max Calls = 2 and Busy Trigger = 2		
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
Set active remote destination of CTIRD1 to be RDD1 (dn = 40822077). User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("40822077", true) on CTIRD1.		
A1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EA3", "9000", "40822077").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 2303 GC1: ConnInProgressEv 2303 GC1: CallCtlConnOfferedEv 2303 GC1: ConnAlertingEv 2303 GC1: CallCtlConnAlertingEv 2303 GC1: TermConnCreatedEv CTIRD1 GC1: TermConnRingingEv CTIRD1 GC1: CallCtlTermConnRingingEv CTIRD1 GC1: ConnConnectedEv 2303 GC1: CallCtlConnEstablishedEv 2303 GC1: TermConnActiveEv CTIRD1 GC1: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9000, CalledAddress = 40822077, CurrentCallingAddress = 9000, CurrentCalledAddress = 2303

Action	Events	Call Info
B1 calls RDD1. User1 invokes Call.connect ("SEP2401C7824EAE", "9001", "40822077"). Call is offered on CTIRD1 and then answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9001 GC2: ConnConnectedEv 9001 GC2: CallCtlConnInitiatedEv 9001 GC2: TermConnCreatedEv SEP2401C7824EAE GC2: TermConnActiveEv SEP2401C7824EAE GC2: CallCtlTermConnTalkingEv SEP2401C7824EAE GC2: CallCtlConnDialingEv 9001 GC2: CallCtlConnEstablishedEv 9001 GC2: ConnCreatedEv 2303 GC2: ConnInProgressEv 2303 GC2: CallCtlConnOfferedEv 2303 GC2: ConnAlertingEv 2303 GC2: CallCtlConnAlertingEv 2303 GC2: TermConnCreatedEv CTIRD1 GC2: TermConnRingingEv CTIRD1 GC2: CallCtlTermConnRingingEv CTIRD1 GC1: CallCtlTermConnHeldEv CTIRD1 GC2: ConnConnectedEv 2303 GC2: CallCtlConnEstablishedEv 2303 GC2: TermConnActiveEv CTIRD1 GC2: CallCtlTermConnTalkingEv CTIRD1	CallingAddress = 9001, CalledAddress = 40822077, CurrentCallingAddress = 9001, CurrentCalledAddress = 2303
CTIRD makes outgoing call to 2302. User1 invokes Call.connect ("CTIRD1", "2303", "2302"). Fail to make outgoing call since max calls limit reached so throws exception.		Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_MAXCALL_ LIMIT_REACHED
Revert back to orig values. Set Max Calls = 4 and Busy Trigger = 2		

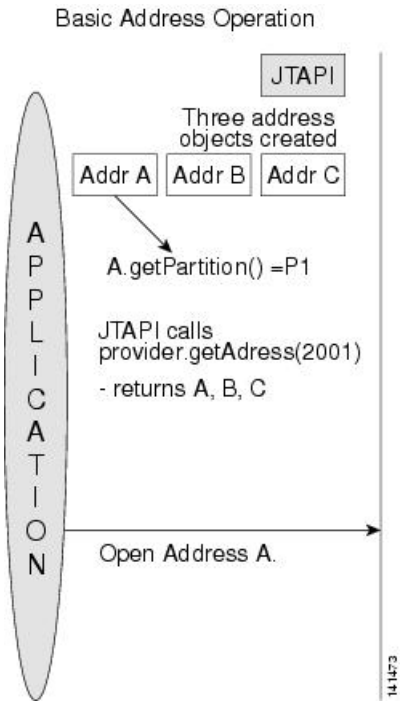
Partition Support

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

Using getPartition() API

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

Using getPartition() API



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

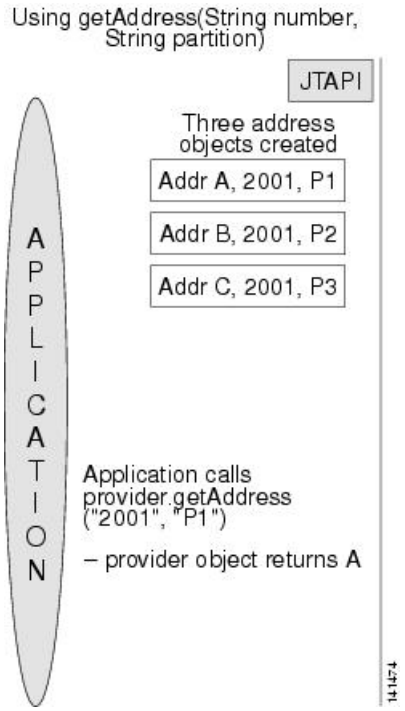
When JTAPI calls provider.getAddress("2001"), the provider object will return an array of three address objects containing A, B and C, since all of them have the same DN.

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

Using getAddress (String Number String Partition)

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

Using getAddress (String Number, String Partition)



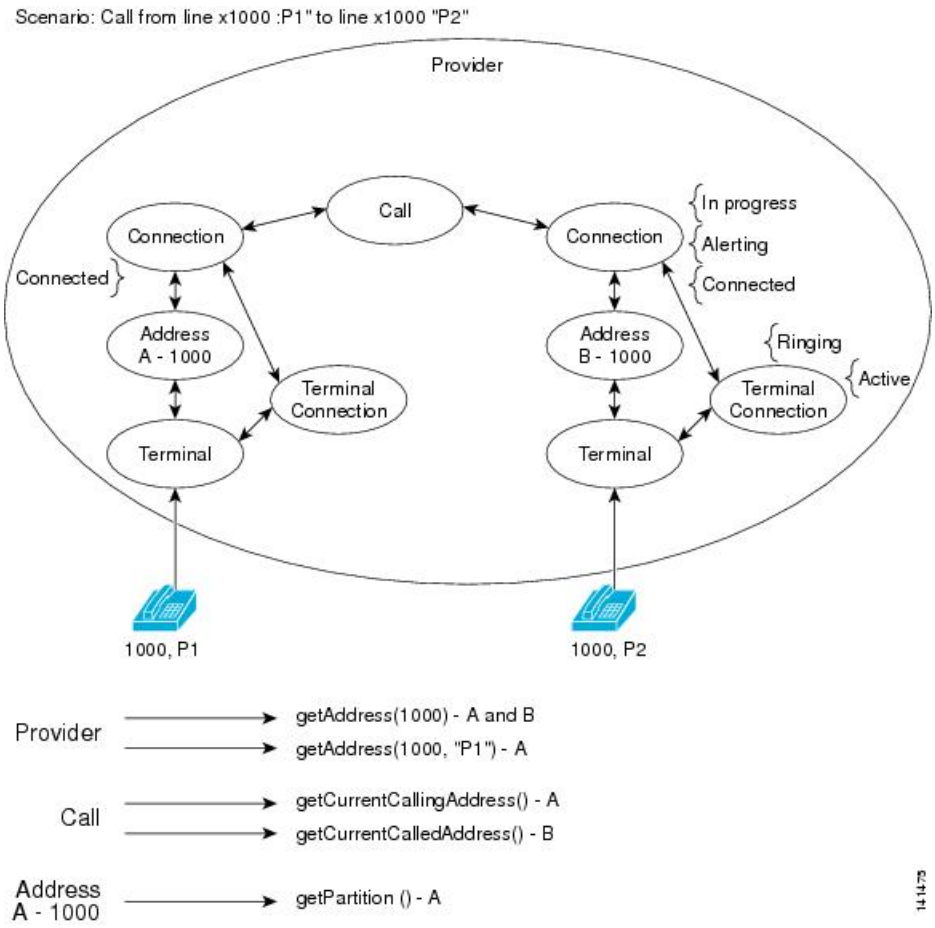
In this case, there are three addresses which belong to three different partitions. Let us denote them by A(2001, P1), B(2001, P2) and C(2001, P3), where 2001, indicates DN and P1, P2, P3 denote different partitions.

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

Simple Call Scenario

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

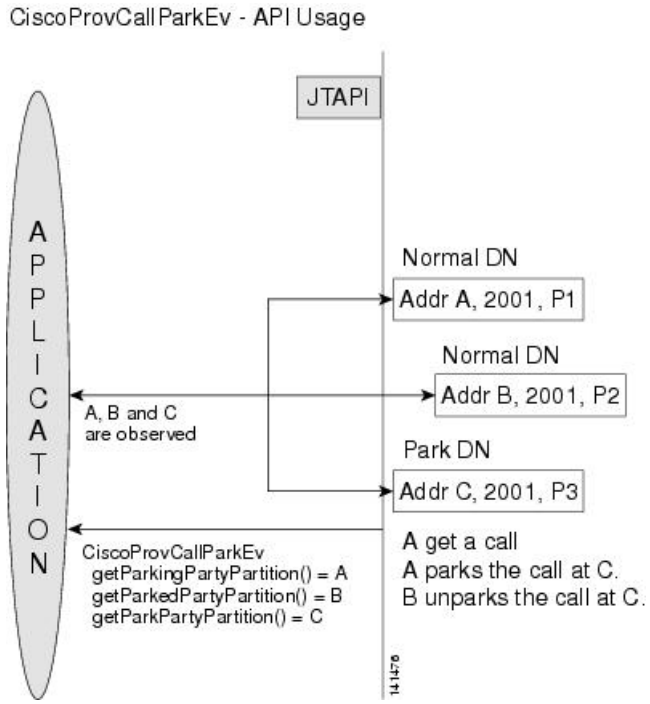
Figure 1: Simple Call Scenario



Park DN

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

Park DN Scenario



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

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

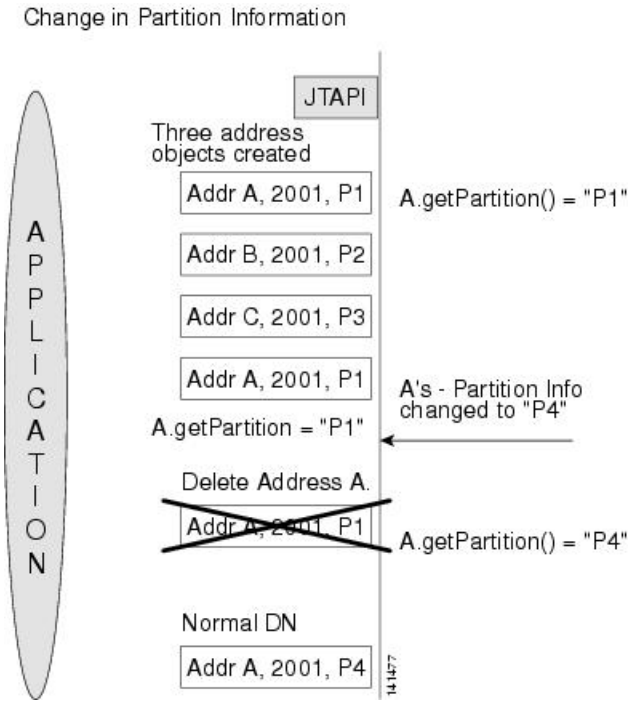
A receives a call and parks the call at C. B unparks the call. While the call is parked, and unparked, CiscoProvCallParkEv is generated. The API

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

Partition Change

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

Change in Partition



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

The new address object will have the new partition information.

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

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

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

JTAPI Partition Support

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

S.No.	Pre-Condition	Scenario	Expected Behavior	Result
1	A and B are two addresses in the same cluster with same DN and different partitions (P1, P2) and in same cluster. A calls B. CSS of A has B's partition in it first.	Calls between addresses with same DN in different device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B
2	A and B are two addresses with same DN in same device but different partitions (P1, P2) and in same cluster. A calls B. CSSof A has B's partition in it first.	Calls between addresses with same DN in same device but different partitions should go through.	Two address objects are created, one for A and one for B. All the appropriate call related events are delivered to both the addresses.	Call is established between A and B.
3	A, B, and C are three different addresses with different DN. (P1, P2, P3). A park DN is configured with same DN as C and different partition (P4).	A calls B. B parks the call. C unparks the call from a park DN which is same as C's DN.	JTAPI should allow C to unpark the call from park DN.	C is successfully able to unpark the call from park DN.
4	A, B, and C are three different addresses having the same DN and different partitions (P1, P2, P3). Apark DN is configured with same DN but belonging to a different partition (P4)	A calls B, B parks call at park DN. C unparks the call.	JTAPI should allow C to unpark the call from park DN.	A is able to call B. B should be able to park the call at the park DN. C should be able to pick up the call.
5	A, B, and C are three different addresses with same DN and different partitions (P1, P2, and P3)	JTAPI calls getPartitionAddress(DN of A).	Three address objects are returned each corresponding to A, B and C.	JTAPI maintains the address objects based on partition info and DN.
6	A and B are addresses with same DN but belong to different partitions (P1, P2).	Application calls getPartition() on the address objects of A and B.		Partition strings of the addresses are returned correctly.
7	A and B are two different addresses (different DNs) belonging to different partitions. A and B are in the control list of the same user. Provider open is completed and the lines are opened.	JTAPI supports old API to open lines when DN is different. There will be no change in behavior.	Lines A and B will be opened, but since they have different DN, user need not specify the partition info. DN alone is sufficient to open the lines, but users can also give partition info and both modes of opening lines will be supported by JTAPI.	Address objects for A and B are created successfully.

S.No.	Pre-Condition	Scenario	Expected Behavior	Result
8	A is an address in the control list of user and is opened by the user. From the CM admin page the partition information of A is changed. The device is restarted after this.	Partition information of a DN is changed. JTAPI should reflect new partition information.	JTAPI will delete the current address object and create a new address object for A when it comes in service again. By querying the partition info of this address object, user should get changed partition info.	New partition information is reflected in the new address object.

Persistent Connection Use Cases

The following pre-conditions apply to all persistent call use cases, unless specified:

- The provider is in IN_SERVICE state.
- All addresses and terminals are already in service.
- Device A (CTI Remote Device - Name: "CTIRDtapi", Line A1 (dn: 881000))

  Remote destination 1 (Name: "rd", Number: "78000")
- Device B (IP Phone - Name: "SEP001319ACCA26", Line B1 (dn: 1000))
- Device C (IP Phone - Name: "SEP00156247EE60", Line C1 (dn: 2000))
- User1 has in its control list: Devices A, B and C. All devices and lines are observed.

Table 45: Call createPersistentCall() on an Address That Is Not Configured to a Remote Terminal Device, i.e. on an IP Phone

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall ("SEP00156247EE60", "5000", "remote") on device C.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED _ON_DEVICE.

Table 46: Call createPersistentCall()on an Address That Is Configured to a Remote Terminal Device Where Active RD Is Not Set

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	Caught exception com.cisco.jtapi.PlatformException: The active remote destination is not set.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_REMOTE_DEVICE_REQUEST _FAILED_ACTIVE_RD_NOT_SET

Table 47: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD Is Set; Verify That Persistent Call Is Connected

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal .setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Action	Events	Call Info
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
User1 invokes CiscoAddress. getPersistentConnection ("CTIRDjtapi") and verify that the connection for the persistent call is returned and uses that to get the Call object and confirm it is for the persistent call.		((CiscoAddress.getPersistentConnection ("CTIRDjtapi")).getCall()).isPersistentCall() = true.
User1 invokes Provider.getCalls()		Provider.getCalls() = null
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A = null
User1 invokes Terminal. getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A = null
Disconnect/drop the persistent call. User1 invokes either Call.drop() or Connection.disconnect()	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 48: Call createPersistentCall() on an Address Configured to a Remote Terminal Device Where a Persistent Call Already Exists

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "6000", "remote2") on device A.	Caught exception com.cisco.jtapi.PlatformException: Persistent Call exists.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_PERSISTENT_CALL_EXISTS.

Table 49: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active Rd Is Set; Verify That Persistent Call Is Connected and Then Have Remote Destination Hang Up

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Remote destination with dn = 78000 hangs up.	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 50: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True; Verify That Persistent Call Is Connected; Set Active RD = False and Verify That Persistent Call Is Dropped

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", false) on device A.	CiscoProvTerminalRemote DestinationChangedEv See persistent call gets dropped: GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = false

Table 51: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True; Verify That Persistent Call Is Connected; Make Incoming Customer Call to Same Remote Terminal Device

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = true.

Actions	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Call.connect("SEP001319ACCA26", "1000", "8881000")	GC2: CallActiveEv GC2: ConnCreatedEv 1000 GC2: ConnConnectedEv 1000 GC2: CallCtlConnInitiatedEv 1000 GC2: TermConnCreatedEv SEP001319ACCA26 GC2: TermConnActiveEv SEP001319ACCA26 GC2: CallCtlTermConnTalkingEv SEP001319ACCA26 GC2: CallCtlConnDialingEv 1000 GC2: CallCtlConnEstablishedEv 1000 GC2: ConnCreatedEv 8881000 GC2: ConnInProgressEv 8881000 GC2: CallCtlConnOfferedEv 8881000 GC2: ConnAlertingEv 8881000 GC2: CallCtlConnAlertingEv 8881000 GC2: TermConnCreatedEv CTIRDjtapi GC2: TermConnRingingEv CTIRDjtapi GC2: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 1000, CalledAddress = 8881000, CurrentCallingAddress = 1000, CurrentCalledAddress = 8881000
Call is answered at device A	GC2: ConnConnectedEv 8881000 GC2: CallCtlConnEstablishedEv 8881000 GC2: TermConnActiveEv CTIRDjtapi GC2: CallCtlTermConnTalkingEv CTIRDjtapi	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", false) on device A.	CiscoProvTerminalRemote DestinationChangedEv Both persistent call with GC1 and customer call with GC2 are not dropped/disconnected even though active rd = false.	A.getActiveRemoteDestinations() = CiscoRemoteDestination Info[1]. CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "78000" CiscoRemoteDestination Info[0].getIsActiveRD() = false.

Actions	Events	Call Info
Customer call with GC2 is disconnected/dropped. User1 invokes either Call.drop() or Connection.disconnect() on the call with GC2.	GC2: TermConnDroppedEv SEP001319ACCA26 GC2: CallCtlTermConnDroppedEv SEP001319ACCA26 GC2: ConnDisconnectedEv 1000 GC2: CallCtlConnDisconnectedEv 1000 GC2: TermConnDroppedEv CTIRDjtapi GC2: CallCtlTermConnDroppedEv CTIRDjtapi GC2: ConnDisconnectedEv 8881000 GC2: CallCtlConnDisconnectedEv 8881000 GC2: CallInvalidEv Since there are no active calls on device A and active rd is now false, the persistent call with GC1 is now dropped/disconnected. GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 52: Have a Persistent Call and Customer Call Connected; Invoke hold() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		

Actions	Events	Call Info
Invoke hold() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON _PERSISTENT _CALL.

Table 53: Have a Persistent Call and Customer Call Connected; Invoke startRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke startRecording() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON_ PERSISTENT _CALL.

Table 54: Have a Persistent Call and Customer Call Connected; Invoke stopRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke stopRecording() on the persistent call with GC1. Make sure Selective call recording is enabled.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT _ALLOWED_ON _PERSISTENT _CALL.

Table 55: Have a Persistent Call and Customer Call Connected; Invoke conference() on the Persistent Call Where Persistent Call Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Actions	Events	Call Info
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where persistent call with GC1 is the primary call and customer call with GC2 is the secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 56: Have a Persistent Call and Customer Call Connected; Invoke conference() on the Persistent Call Where Persistent Call Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where customer call with GC2 is primary call and persistent call with GC1 is secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 57: Have a Persistent Call and Customer Call Connected; Invoke park() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke park().	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 58: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(Call) where persistent call with GC1 is primary call and customer call with GC2 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 59: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Primary to Another DN Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(String address) where persistent call with GC1 is primary call to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 60: Have a Persistent Call and Customer Call Connected; Invoke transfer() on the Persistent Call Where PC Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		

Actions	Events	Call Info
Invoke transfer(Call) where customer call with GC2 is primary call and persistent call with GC1 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 61: Have a Persistent Call and Customer Call Connected; Invoke consult() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Make consult call from device A to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 62: Have a Persistent Call and Customer Call Connected; Invoke pickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke pickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 63: Have a Persistent Call and Customer Call Connected; Invoke otherPickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke otherPickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Table 64: Have a Persistent Call and Customer Call Connected; Invoke redirect() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke redirect("2000") on the persistent call.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION _NOT_ALLOWED _ON_PERSISTENT _CALL.

Play Announcement

Prerequisites

Pre-conditions to all play announcement use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Device A (CTI Remote Device - Name: "CTIRD3", Line A1 (dn: 9202))
    - o Remote destination 1 (Name: "C1_CTIRD3_RDD1", Number: "339006")
- Device B (IP Phone - Name: "SEP2401C7824EA3", Line B1 (dn: 9000))

- Announcement Identifier is Welcome Greeting Sample.
- User1 has in its control list: Devices A, and B. All devices and lines are observed.

Basic Play Announcement Use Cases

Basic Play Announcement Use Cases

Table 65: Play Announcement on CTI Remote Device with Persistent Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnOfferedEv Unknown	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
User1 invokes Provider.getCalls()		Provider.getCalls() returns the announcement call.
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for the announcement call.

Table 66: Play Announcement That Stopped Playing Before the End of the Announcement

Action	Events	Call info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call info
User1 invokes Provider.getCalls()		Provider.getCalls() returns the announcement call.
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for the announcement call.
Disconnect/drop the announcement call. User1 invokes either Call.drop() or Connection.disconnect() to stop the announcement before it finishes playing.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 67: Play Announcement with Incoming Customer Call in Ringing State

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
User1 invokes Provider.getCalls()		Provider.getCalls() returns both the customer call and the announcement call
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A returns the Connection for both the customer call and the announcement call.
User1 invokes Terminal.getTerminalConnections() on device A.		Terminal.getTerminalConnections() on device A returns the TerminalConnection for both the customer call and the announcement call.

Action	Events	Call Info
After announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 68: Play Announcement Where the Call Is Answered Before the Full Message Plays

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	

Action	Event	Call Info
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
Customer call is answered. Announcement call is dropped/disconnected.	GC3: CallCtlTermConnHeldEv CTIRD3 GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 69: Play Announcement Where the Call Is Answered Before the Full Message Plays

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is left ringing.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	

Action	Event	Call Info
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
Customer call is answered. Announcement call is dropped/disconnected.	GC3: CallCtlTermConnHeldEv CTIRD3 GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 70: Play Announcement with Call in Connected State

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not available in current state.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. OPERATION_NOT_AVAILABLE_ IN_CURRENT_STATE.

Table 71: Play Announcement with Held Customer Call

Action	Event	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Event	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Event	Call Info
User1 invokes Call.connect("SEP2401C7824EA3", "9000", "9202") and is answered.	GC2: CallActiveEv GC2: ConnCreatedEv 9000 GC2: ConnConnectedEv 9000 GC2: CallCtlConnInitiatedEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnActiveEv SEP2401C7824EA3 GC2: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC2: CallCtlConnDialingEv 9000 GC2: CallCtlConnEstablishedEv 9000 GC2: ConnCreatedEv 9202 GC2: ConnInProgressEv 9202 GC2: CallCtlConnOfferedEv 9202 GC2: ConnAlertingEv 9202 GC2: CallCtlConnAlertingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnRingingEv CTIRD3 GC2: CallCtlTermConnRingingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalking CTIRD3	
User1 puts the customer call on hold. Invokes TerminalConnection.hold() on Device A.	GC2: CallCtlTermConnHeldEv CTIRD3	

Action	Event	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
After the announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 72: Play Announcment with an Outgoing Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202

Action	Events	Call Info
Device A makes outgoing call to Device C. User1 invokes Call.connect("CTIRD3", "9202", "9000"). Call is left unanswered (ringing state).	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: ConnConnectedEv 9202 GC2: CallCtlConnInitiatedEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: CallCtlConnDialingEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv 9000 GC2: ConnInProgressEv 9000 GC2: CallCtlConnOfferedEv 9000 GC2: ConnAlertingEv 9000 GC2: CallCtlConnAlertingEv 9000 GC2: TermConnCreatedEv SEP2401C7824EA3 GC2: TermConnRingingEv SEP2401C7824EA3 GC2: CallCtlTermConnRingingEv SEP2401C7824EA3	

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: TermConnDroppedEv SEP2401C7824EA3 GC2: CallCtlTermConnDroppedEv SEP2401C7824EA3 GC2: ConnDisconnectedEv 9000 GC2: CallCtlConnDisconnectedEv 9000 GC2: CallInvalidEv GC2: CallObservationEndedEv GC3: CallActiveEv GC3: ConnCreatedEv 9202 GC3: CallCtlConnDialingEv 9202 GC3: TermConnCreatedEv CTIRD3 GC3: TermConnActiveEv CTIRD3 GC3: CallCtlTermConnTalkingEv CTIRD3 GC3: ConnConnectedEv 9202 GC3: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample
	GC3: ConnCreatedEv Unknown GC3: ConnInProgressEv Unknown GC3: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC3: ConnConnectedEv Unknown GC3: CallCtlConnEstablishedEv Unknown	
	GC3: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call Info
After announcement finishes playing, call is disconnected.	GC3: CiscoAnnouncementEndedEv GC3: TermConnDroppedEv CTIRD3 GC3: CallCtlTermConnDroppedEv CTIRD3 GC3: ConnDisconnectedEv Unknown GC3: CallCtlConnDisconnectedEv Unknown GC3: ConnDisconnectedEv 9202 GC3: CallCtlConnDisconnectedEv 9202 GC3: CallInvalidEv GC3: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error

Table 73: Play Announcement on an IP Phone

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("SEP8478ACE7F9B9", "Welcome Greeting Sample") on ip phone.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED_ ON_DEVICE.

Table 74: Play Announcement on the CTI Port

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("CTI3", "Welcome Greeting Sample") on CTI Port.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED_ ON_DEVICE.

Table 75: Play Announcement on CTI Remote Device Without Persistent Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: No persistent call exists.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_NO_PERSISTENT_ CALL_EXISTS.

Table 76: Play Announcement on a CTI Remote Device While a Persistent Call Is Being Set Up

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
Call is offered to the remote destination but not picked up.		
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Persistent Call is being set up.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_PERSISTENT_CALL_ BEING_SETUP.

Table 77: Play Announcement on CTI Remote Device with a Persistent Call with an Invalid Announcement Identifier

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Invalid Media resource ID.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_INVALID_MEDIA_ RESOURCE_ID.

Table 78: Play Announcement Back to Back

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID() = Welcome Greeting Sample CiscoAnnouncementEndedEv. getSuccess() = true. CiscoAnnouncementEndedEv. getErrorCode() = 0 CiscoAnnouncementEndedEv. getErrorDescription() = No Error
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	For these 3 call events, feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv.	CiscoAnnouncementStartedEv. will have feature reason = CiscoFeatureReason. REASON_PLAY_ANNOUNCEMENT

Action	Events	Call Info
After announcement finishes playing, the call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	Caught exception com.cisco.jtapi.PlatformException: Announcement already in progress.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_ANNOUNCEMENT_ALREADY_IN_PROGRESS.

Table 79: Play Announcement to Stop IPVMS Service

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
Stop IPVMS service on all the nodes.		

Action	Events	Call Info
User1 invokes CiscoAddress. startAnnouncement("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202 GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown GC2: CiscoAnnouncementErrorEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementErrorEv. getErrorCode() = -1932787536 CiscoAnnouncementErrorEv. getErrorDescription() = Resource Not Available.

Play Announcement Feature Interaction Use Cases

Play Announcement Feature Interaction Use Cases

Table 80: Hold Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes TerminalConnection.hold () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Table 81: Redirect Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes Connection.redirect () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Table 82: Park Announcement Call

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoAddress.createPersistentCall ("CTIRD3", "1000", "remote") on device A and remote destination answers.	GC1: CallActiveEv GC1: ConnCreatedEv 9202 GC1: ConnInProgressEv 9202 GC1: CallCtlConnOfferedEv 9202 GC1: ConnCreatedEv 1000 GC1: ConnConnectedEv 1000 GC1: CallCtlConnEstablishedEv 1000 GC1: ConnAlertingEv 9202 GC1: CallCtlConnAlertingEv 9202 GC1: TermConnCreatedEv CTIRD3 GC1: TermConnRingingEv CTIRD3 GC1: CallCtlTermConnRingingEv CTIRD3 GC1: ConnConnectedEv 9202 GC1: CallCtlConnEstablishedEv 9202 GC1: TermConnActiveEv CTIRD3 GC1: CallCtlTermConnTalkingEv CTIRD3	CallingAddress = 1000, CalledAddress = 9202, CurrentCallingAddress = 1000, CurrentCalledAddress = 9202
User1 invokes CiscoAddress.startAnnouncement ("CTIRD3", "Welcome Greeting Sample") on Device A.	GC2: CallActiveEv GC2: ConnCreatedEv 9202 GC2: CallCtlConnDialingEv 9202 GC2: TermConnCreatedEv CTIRD3 GC2: TermConnActiveEv CTIRD3 GC2: CallCtlTermConnTalkingEv CTIRD3 GC2: ConnConnectedEv 9202 GC2: CallCtlConnEstablishedEv 9202	CiscoAnnouncementStartedEv. getAnnouncementID () = Welcome Greeting Sample
	GC2: ConnCreatedEv Unknown GC2: ConnInProgressEv Unknown GC2: CallCtlConnOfferedEv Unknown	Feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT
	GC2: ConnConnectedEv Unknown GC2: CallCtlConnEstablishedEv Unknown	
	GC2: CiscoAnnouncementStartedEv	CiscoAnnouncementStartedEv will have feature reason = CiscoFeatureReason. REASON_PLAY_ ANNOUNCEMENT

Action	Events	Call Info
User1 invokes Connection.park () on the announcement call.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode () = CiscoJtapiException. CTIERR_OPERATION_ NOT_ALLOWED.
After announcement finishes playing, call is disconnected.	GC2: CiscoAnnouncementEndedEv GC2: TermConnDroppedEv CTIRD3 GC2: CallCtlTermConnDroppedEv CTIRD3 GC2: ConnDisconnectedEv Unknown GC2: CallCtlConnDisconnectedEv Unknown GC2: ConnDisconnectedEv 9202 GC2: CallCtlConnDisconnectedEv 9202 GC2: CallInvalidEv GC2: CallObservationEndedEv	CiscoAnnouncementEndedEv. getSuccess () = true. CiscoAnnouncementEndedEv. getErrorCode () = 0 CiscoAnnouncementEndedEv. getErrorDescription () = No Error

Play Zip Tone

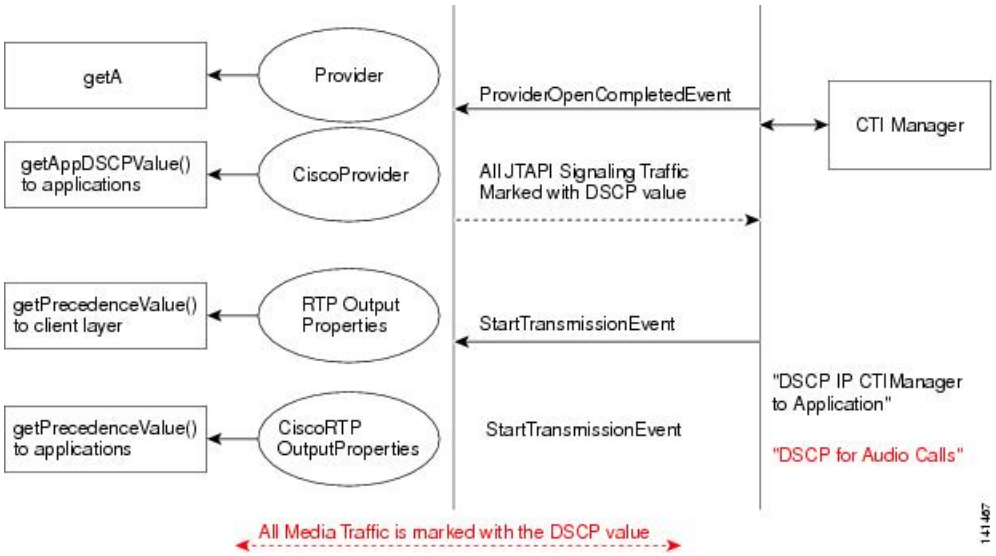
A and B represents the terminals. TermConnB represents the Cisco terminal connection of Terminal B. TermConnCTI1 represents the Cisco terminal connection of terminal CTI1. CTI1 is a Cisco media terminal.

Sl.No	Scenario	Events/Response
1.	A calls B. B answers the call and the application invokes TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	User on B hears the ZIP tone.
2.	A calls B. B answers the call and application invokes TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	User on A hears the ZIP tone.
3.	A calls B. B is alerting. Application calls TermConnB. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	Tone is heard by user B.
4.	A calls CTI1. Call is answered by CTI1. Application calls TermCTI. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	PlatformException is thrown to application request.

Sl.No	Scenario	Events/Response
5.	A calls CTI1. Call is answered by CTI1. Application calls TermConnCTI1. playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	User on A hears the ZIP tone.
6.	A, B and CTI1 are in conference. Application calls TermConnB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_LOCALONLY)	User on B hears the ZIP tone.
7.	A, B and CTI1 are in conference. Application calls TermB.playTone(CiscoTone.ZIPTONE, CiscoCall. PLAYTONE_REMOTEONLY)	None of the users on A, B and CTI hear the tone.

QoS Support

Figure 2: Call Flow Diagram for QoS Support



JTAPI QoS

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

1. Start Registry Editor (Regedt32.exe).
2. Go to the following key:

HKEY_LOCAL_ MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\

Note The registry key is one path.

1. On the Edit menu, click Add Value.
2. Type DisableUserTOSSetting.
3. Click REG_DWORD in the Data Type box.
4. Click OK.
5. Enter 0 in the prompt box.
6. Quit the Registry Editor.
7. Restart the computer.

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

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

QSIG Path Replacement

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

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

No.	Action	Event
1	A registered with CM1, B is registered with CM2, and C registered with CM3. A calls B (GC1); B transfers the call to C. The application is monitors C. The PR feature replaces the path after the call gets connected to C. The same action applies to scenarios that involve call forward at B. (The called party transfers the call.)	These events get delivered to applications: CallCtrlConnectionEstablishedEv A CallCtrlConnectionDisConnectEv B OpenLogicalChannelEvent if C is a CTI device (Dynamically registered CTIPorts and RP)

No.	Action	Event
2	A registered with CM1, B registered with CM2, and C registered with CM3. B calls C; C answers; B transfers the call to A. A answers. The application is monitors only C. (The calling party transfers the call.)	In this case, both A are C represent called parties when transfer completes. After the call is answered, PR replaces the path. In this case, A and C represent IP phones; the display will be updated as a part of PR feature operation, that makes either A or C as calling. JTAPI events: GC1: CallCtlConnEstablishedEv A GC1: CallCtlConnDisconnectedEv B
3	Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. A calls B (GC1); B answers and transfers the call to C (GC2). Path replacement connects A and C bypassing CM2. The application observes both A and C. (The called party transfers the call.)	For GC1 Call Observer: GC1: CallCtlConnEstablishedEv C GC1: CallCtlConnDisconnected B Before the PR feature replaces the path, the application sees two calls: GC1 with connections to A and C (external) and GC2 with connections to C and A (external). When the PR feature replaces the path, either GC1 changes GC2, or GC2 changes to GC1. Assuming A's GCID changes from GC1 to GC2: GC1: CiscoCallChangedEv (oldGCID = GC1, newGCID = GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: TermConnTalkingEvent for TerminalA cause = CAUSE_QSIG_PR

No.	Action	Event
4	Trombone case: A registered to CM1, B registered to CM2, and C registered to CM1. B calls A and transfers the call to C. Path replacement connects A and C, bypassing CM2. Application observes both A and C. (The calling party transfers the call.)	Before the PR feature replaces the path, the application will see two calls: GC1 with connections to A and B (external) and GC2 with connections to C and B (external). In this case, the application will not see any transfer start events. When PR feature replaces the path, it updates the display of A and C and path gets replaced, resulting in a GCID change. Assuming A's GCID is changed and made the calling party, the following JTAPI events occur: GC1: CiscoCallChangedEv (GC1 to GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: ConnCreatedEv A GC2: ConnConnectedEv A GC2: TermConnTalkingEvent for TerminalA cause = CAUSE_QSIG_PR
5	A registered to CM1, B registered to CM2, and C registered to CM1. A calls B; B transfers the call to C. C answers and before path replacement completes, C invokes a feature (park, redirect, and so on).	Path replacement gets abandoned.
6	In some conditions, call processing ignores feature requests (redirect, park, transfer, and so on). This happens when a setup request is sent out, and the local CM is waiting for connect from the other end.	JTAPI: Exception will be thrown from JTAPI for feature requests.
7	In some cases, the application could receive dead air when CM goes down when the PR feature is trying to switch the RTP path. This could happen to a previously connected call.	No events JTAPI Apps: Hang up the call

Recording Use Cases

Expose ClusterID in CiscoProvider Interface

Actiions	Events	Call Info
Start application and initialize JTAPI		
Add provider observer	ProvInServiceEv	
Application calls ciscoProvider.getClusterID()		JTAPI returns 'StandAloneCluster'

Admin changes the clusterID to '3NodeClusterinSanJose' and restarts the services		
	ProvOutofService ProvInServiceEv	Application calls getClusterID() - JTAPI returns ' 3NodeClusterinSanJose'

Multi-Clusters Gateway Recording Use Cases

Test configurations Specification:

- Cluster 1 = SME Cluster with 1 PUB and 1 SUB (Note: Only use DN's 2xxx, 3xxx, 9xxx)
- Cluster 2 = External Cluster (Note: Only use DN's 1xxx, 4xxx)
- Cluster 3 = Leaf Cluster (Note: Only use DN's 5xxx, 6xxx)

Assumption: All devices have BIB enabled unless specified in the detailed test cases. CTIRD does not support BIB.

- Cluster 1 and Cluster 2 are connected thru SIP GW which is recording enabled.
- Cluster 1 and Cluster 3 are connected thru SIP trunk ICT.
- Cluster 2 and Cluster 3 are connected thru SIP trunk ICT.

On Cluster 1 Route-Patterns:

- 180.XXXX: SIP ICT trunk from cluster 1 to cluster 3, calledPartyTranformation: remove PreDot.
- 171.XXXX: SIP GW from cluster 1 to cluster 2, calledPartyTranformation: remove PreDot.

On Cluster 2 Route-Patterns:

- 172.XXXX: SIP GW from cluster 2 to cluster 1, calledPartyTranformation: remove PreDot.
- 172180.XXXX: SIP GW from cluster 2 to cluster 1 to cluster 3, calledPartyTranformation: remove PreDot.
- 180.XXXX: SIP ICT trunk from cluster 2 to cluster 3 directly, calledPartyTranformation: remove PreDot.

On Cluster 3 Route-Patterns:

- 172.XXXX: SIP ICT trunk from cluster 3 to cluster 1, calledPartyTranformation: remove PreDot.
- 172171.XXXX: SIP GW from cluster 3 to cluster 1 to cluster 2, calledPartyTranformation: remove PreDot.
- 171.XXXX: SIP ICT trunk from cluster 3 to cluster 2 directly, calledPartyTranformation: remove PreDot.

Recording IP Phones

Scenario 1

IP phones (Basic): Multi-Clusters Gateway preferred with auto recording

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP8E0E) has line B1:4006 • GW preferred • auto recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEP3B5F) has line A1 3601 configured as: • GW preferred • auto recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** Verify A1 have recording type AUTO_RECORDING **4.** A1 (cluster3) call B1 (cluster2) (e.g. 3602 call 1721714006) **5.** B1 answers the call **6.** A1 drops	Step 3- Check A1 and B1 CiscoAddres.AUTO_RECORDING Step 5- Check two recording sessions established on A1 and B1. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Check B1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = recorder B device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of B .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster2 Step 6- Check A1 and B1 get CiscoTermConnRecordingEndEv and disconnect on recorders.

Scenario 2

IP phones (Basic): Multi-Clusters Gateway preferred with auto recording

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP8E0E) has line B1:4006 • GW preferred • auto recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEP3B5F) has line A1 3601 configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** Verify A1 have recording type SELECTIVE_RECORDING **4.** A1 calls B1 (3601 calls 1721714006) **5.** B1 answers **6.** A1 requests startRecording(app) **7.** A1 requests stopRecording() **8.** A1 disconnects B1	Step 3- Check A1 CiscoAddres.SELECTIVE_RECORDING Step 5- Check auto recording started on B1. Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check B1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = recorder B device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of B .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster2 Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check A1 get CiscoTermConnRecordingEndEv and disconnect on recorders.

Scenario 3

IP phones (Basic): Multi-Clusters Gateway preferred with press key invoke recording

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP723F) has line B1:4008 • GW preferred • selective recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEPDB17) has line A1 2303 configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** Verify A1 have recording type SELECTIVE_RECORDING **4.** A1 calls B1 (2303 calls 1721714008) **5.** B1 answers **6.** Press recording key on A **7.** A1 disconnects B1	Step 3- Check A1 and B1 CiscoAddres.SELECTIVE_RECORDING Step 6-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_DEVICE .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1

Scenario 4

IP phones (Basic): Multi-Clusters Gateway preferred with selective recording and Cluster 1 fork media to branch recorder on cluster3

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP723F) has line B1:4008 • GW preferred • selective recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEPDB17) has line A1 1623 configured as: • GW preferred • selective recording enabled • recording profile:rec_profile2: 1802000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** A1 call B1 (1623 call 1721714008) **4.** B1 answer the call **5.** Start silent recording on A1 **6.** Stop recording on A1 **7.** A1 drop	Step 5- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A2 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT .getTerminalName() = SIPICT-To-Cluster3 .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk name – SIP GW.getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 6- Check A1 get CiscoTermConnRecordingEndEv

Scenario 5

IP phones (Basic): Multi-Clusters Device preferred with selective recording and device fork media to central recorder

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
• Phone B (SEP723F) has line B1:4008
  • selective recording enabled
Cluster 3:
• Branch recorder: 2000
• Phone A (SEPDB17) has line A1 2008 configured as:
  • Device preferred
  • selective recording enabled
  • recording profile:rec_profile2: 1722000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (A1 call 70662050
4. B1 answer the call
5. A1 request startRecording(app)
6. A1 request stopRecording()
7. A1 drop
	Step 5-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 1722000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE
	.getMediaForkingDeviceName() = device name of A
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster3
	Step 6- Check A1 get CiscoTermConnRecordingEndEv and disconnect on recorder.

Scenario 6

IP phones (Basic): Multi-Clusters Hold and resume - Gateway preferred with automatic recording

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
• Phone B (SEP8E0E) has line B1:4006
  • GW preferred
  • auto recording enabled
  • recorder B:2000
Cluster 3:
• Branch recorder: 2000
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile1: 2000
• Phone A (SEP3B5F) has line A1 3602 configured as:
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (3602 call 1721714006)
4. B1 answer the call
5. A1 put call on hold
6. A1 resume the call
7. A1 drop
	Step 4-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name – SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = recorder B device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk 2 device name – SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster2
	Step 5- Check A1 get CiscoTermConnRecordingEndEv
	Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Step 7- Check A1 get CiscoTermConnRecordingEndEv

Scenario 7

IP phones (Basic): Hold and resume - Phone preferred with Selective Recording

Phone A has line : 1506 Phone B has line : 1504 • Phone Prefered • Selective Recording enabled On Phone B • Recorder Phone C : 1505 • Recorder Phone D : 1503 • recording profile: rec_profile : 1505 **1.** Open provider **2.** Observe A, B, C, D in Prov1 **3.** A call B (1506 calls 1504) **4.** B answer the call. **5.** Selective recording was initiated on Phone B. **6.** B put call on hold. **7.** B resumes the call. **8.** A drop the call.	Step 4-Check B get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check B TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN = 1505) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = Phone Name .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster Check B TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = recorder B device name .getAddress() = Recording profile (DN 1505) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = Phone name .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster Step 6- Check B get CiscoTermConnRecordingEndEv Step 7- Check B get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv Step 8- Check B get CiscoTermConnRecordingEndEv

Scenario 8

IP phones (Basic): Multi-Clusters Multiple calls - Gateway preferred with automatic recording

Setup and Action	Events and Call Info

Setup and Action	Events and Call Info
Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: • Phone B (SEP8E0E) has line B1 1681 • GW preferred • selective recording enabled • recorder B:2000 • Phone C (SEP723F) has line B1: 4008 • GW preferred • selective recording enabled • recorder B:2000 Cluster 3: • Branch recorder: 2000 • Phone A (SEPDB17) has line A1 3602 configured as: • GW preferred • auto recording enabled • recording profile:rec_profile1: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1, C1 in Prov2 **3.** A1 call B1 (3602 call 1721714006) **4.** B1 answer the call **5.** A1 put call on hold **6.** A1 call C1 (3602 call 1721714008) **7.** C1 answer **8.** A1 retrieve first call **9.** A1 retrieve second call **10.** A1 drop second call	Step 4- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recording device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 5- Check A1 get CiscoTermConnRecordingEndEv Step 7-Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. (Recording is started on A1 for call A1->C1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 8- Retrieve first call. Check A1 get CiscoTermConnRecordingEndEv (for A1->C1) Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv (for A1->B1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() =

Setup and Action	Events and Call Info
11. A1 drop first call	.getMediaForkingClusterID() = name of cluster1 Step 9- Retrieve second call. Check A1 get CiscoTermConnRecordingEndEv (for A1->B1) Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv (for A1->C1) Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name – SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1

CTI Remote Devices Use Cases

Scenario 9

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with automatic recording- IP phone (cluster3) call remote device (cluster1)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
• devB (CTIRD3) has line B1:9008 (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
 	
Cluster 2:
• Phone C has line C1 (1681)
 	
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile2: 2000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 (cluster 3) call B1 (cluster1) (3601 call 1729008)
4. Remote destination on cluster 2 answer the call
5. A1 drop
	Step 4- Check auto recording started on A1.
	Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check A1 get CiscoTermConnRecordingEndEv
	Note: Auto recording start on B1 (new changes)
	Step 4-
	Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check B1 get CiscoTermConnRecordingEndEv

Scenario 10

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with silent recording- IP phone (cluster3) call remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
Cluster 2:
• devB (CTI_RD3) has line B1:1625 (remote destination: 1722602 on cluster 1) configured as:
  • selective recording enabled
  • GW preferred
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 call B1 (3601 calls 1721711620)
4. Remote destination on cluster 1 answer the call
5. Start silent recording on A1
6. Start silent recording on B1
7. Stop recording on A1
8. A1 drop
	Step 5- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check A1 get CiscoTermConnRecordingEndEv
	Note: Auto recording start on B1 (new changes)
	Step 6-
	Check B1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check B1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check B1 get CiscoTermConnRecordingEndEv

Scenario 11

CTI Remote Devices (Basic): Multi-Clusters Gateway preferred with automatic recording- Remote device (cluster3) call Remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
Cluster 2:
• devB (remote device CTI_RD2) has line B1:1624 (remote destination: 2303 on cluster 1) configured as:
  • selective recording enabled
  • GW preferred
Cluster 3:
• Branch recorder: 2000
• devA (remote device CTI_RD3) has line A1 1622 (remote destination: 9000 on cluster 1) configured as:
  • GW preferred
  • auto recording enabled
  • recording profile:rec_profile2: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1 in Prov3, B1 in Prov2
3. A1 (cluster 3) call B1 (cluster2)
4. Call is offered to devA's remote destination
5. Remote destination of devA answer
6. Call is offered to B1
7. DevB's remote destination answer the call
8. Remote destination of devB drop (or devB drop)
	Step 7- Check auto recording started on A1.
	Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check A1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1

Scenario 12

CTI Remote Devices (Basic): Multi-Clusters Hold and resume with automatic recording-IP phone call (cluster3) remote device (cluster2)

Setup and Action	Events and Call Info

Cluster 1: • "Allow trunk use GW recording" is enabled • Central recorder: 2000 • Mobile through PSTN GW in Cluster 1 Cluster 2: • devB (CTI_RD3) has line B1:1620 (remote destination: 1722602 on cluster 1) configured as: • auto recording enabled • GW preferred Cluster 3: • Branch recorder: 2000 • devA (IP phone) has line 3602 configured as: • GW preferred • auto recording enabled • recording profile:rec_profile2: 2000 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe A1 in Prov3, B1 in Prov2 **3.** A1 (cluster 3) call B1 (cluster2) **4.** Remote destination of devB answer the call **5.** A1 put call on hold **6.** A1 resume the call **7.** A1 drop	Step 4- Check auto recording started on A1. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 5- Check A1 get CiscoTermConnRecordingEndEv Step 6- Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check A1 get CiscoTermConnRecordingEndEv

Feature Interaction: Recording Use Cases

Scenario 13

FI: Redirect - IP phone (cluster3) call auto recording remote device (cluster1), redirect to IP phone (cluster3)

Setup and Action	Events and Call Info

Cluster 1:
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
• Mobile through PSTN GW in Cluster 1
• devRD (CTIRD3) has line RD1:9008 (active remote destination: 1711681 on cluster2) configured as:
  • GW preferred
  • automatic recording enabled
Cluster 2:
• Phone D (RDD) has line D1 (1681)
Cluster 3:
• Branch recorder: 2000
• Phone A (SEP3B5F) has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• Phone B (SEPDB17) has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe A1, B1 in Prov3, D1 in Prov2, RD1 in Prov1
3. A1 (cluster3) call RD1 (cluster1) (3601 call 1729008)
4. Remote destination on cluster 2 answer the call
5. A1 redirect to B1
6. B1 disconnect the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- Check RD1 get CiscoTermConnRecordingEndEv
	Step 5- Check auto recording restarted on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 14

FI: Redirect -devRD (cluster1/SME), RDD (cluster2), A(cluster3/leaf) and B (cluster3/leaf) - RD auto recording and RD redirect

Setup and Action	Events and Call Info

Cluster 1:
• Central recorder: 2000
• devRD (CTIRD3) has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • automatic recording enabled
  • GW preferred
 	
Cluster 2:
• RDD has line RDD1 (1681)
 	
Cluster 3:
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devA (SEPDB17) has line A1 (2008) configured as:
  • Phone preferred
  • auto recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP3B5F) has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
 	
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1 in Prov1, RDD1 in Prov2, A1, B1 in Prov3
3. A1 (cluster3) call RD1 (cluster1)
4. Remote destination on cluster 2 answer the call
5. RD1 redirect to B1 (cluster3)
6. B1 answer	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 4- Check auto recording started on A1. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = branch recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of A .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster3 Step 5: Check Recording retriggered on A1 Check A1 get CiscoTermConnRecordingEndedEv. Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check A1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = branch recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE .getMediaForkingDeviceName() = device name of A

Setup and Action	Events and Call Info
	.getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster3

Scenario 15

FI: Redirect- devRD (leaf), RDD (SME), A (SME) and B (cluster2) - RD auto recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1729000 on cluster1) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 1.
• devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP723F) on cluster 2 has line B1 (4008) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1 in Prov1, RDD1 in Prov2, A1, B1 in Prov3
3. A1 call RD1
4. Remote destination answer the call
5. A1 redirect to B1
6. B1 answer
7. Close RD1 and reopen RD1 (unobserved and reobserve RD1)
8. RD1 disconnects the call	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 6- Recording retriggered on RD1. Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check RD1 CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() =

Setup and Action	Events and Call Info
	.getMediaForkingClusterID() = name of cluster1 Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 16

FI: Redirect- devRD (SME), RDD (leaf), A (leaf) and B (cluster2) - RD silent recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD4) on cluster 1 has line RD1 (9008) (active remote destination: 1802303 on cluster3) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 3.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP723F) on cluster 2 has line B1 (4008) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start silent recording on RD1
6. A1 redirect to B1 thru GW (i.e. A1 redirect to 171XXXX)
7. B1 answer
8. Start app recording on RD1
9. App stop recording on RD1
	Step 5- Start recording should fail on RD1 (error = CTIERR_RESOURCE_NOT_AVAILABLE??)
	Step 8- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 17

FI: Transfer- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 setup transfer to B1
6. B1 answer
7. A1 complete transfer
8. B1 disconnects the call
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Complete Transfer. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 18

FI: Direct Transfer- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A direct transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 call B1
6. B1 answer
7. App send direct transfer request on A1 with primary call = A1-RD1
8. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 19

FI: Conference- devRD (leaf), RDD (cluster2), A (SME) and B (SME) - RD silent recording and A conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start silent recording on RD1
6. A1 setup conference to B1
7. B1 answer
8. A1 complete conference
9. Close line RD1 and reopen line RD1
10. App stop recording on RD1
11. RD disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete conference. A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1.
	Step 9- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 10- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 20

FI: Join calls - devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A join calls

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 call A1
6. A1 answer
7. A1 join two calls wit primary call = A1 –RD1 call
8. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1.
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 21

FI: JAL- devRD (SME), RDD (cluster2), A1, A2 and B (leaf) - RD silent recording and A1 does JAL

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) and A2 (1623) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEP3B5F) on cluster 3 has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start silent recording on RD1
6. A2 call B1
7. B1 answer
8. A1 join two calls with primary call = A1 RD1
9. Stop recording on RD1
10. RD1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- A2 and B1 are connected.
	Step 8- JAL. A1, B1 and RD1 in conference. Recording *not* retriggered on RD1.
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 22

FI: Drop any party- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A drop B from conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1 set up conference to B1
6. B1 answer
7. A1 complete conference
8. Reopen RD1
9. A drop B from conference
10. RD1 disconnects the call	

Setup and Action	Events and Call Info
	Step 4- Check auto recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- A1 complete conference. A1, B1 and RD1 are in conference. Recording *not* retriggered on RD1. Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 9- A drop B. A1 and RD1 are connected as two parties call. Recording retriggered on RD1. Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

Setup and Action	Events and Call Info
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 10- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 23

FI: EM- devRD (leaf), RDD (cluster2), A (SME) - RD auto recording and A EM Login

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPAB21) on cluster 1 has line A1 (9000) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• EM profile with line E1 (7788)
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. EM login to devA with line E1
4. E1 call RD1
5. Remote destination answer the call
6. E1 disconnect call
7. EM logout
	Step 5- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- EM logout succeed.
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 24

Hunt list - devRD (leaf), RDD (cluster2), A (SME), B (SME) and HP(SME) - RD selective recording

Setup and Action	Events and Call Info
Cluster 1 (SME): • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: Cluster 3 (leaf): • "Allow trunk use GW recording" is enabled • Branch recorder: 2000 • devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as: • selective recording enabled • GW preferred Remote Destination RDD on cluster 2. • devA (IP9) on cluster 1 has line A1 (2302) configured as: • GW preferred • selective recording enabled • recording profile:rec_profile1: 2000 • devB (RP) on cluster 1 has line B1 (1500) configured as: • no recording • Cluster 1 (SME): • Line Group: LG2: A1 and B1, top down • Hunt list: HL2:LG2 • Hunt pilot: 3636 - HL2 **1.** Open provider Prov1, Prov2, Prov3 **2.** Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly **3.** RD1 call HP 3636 (1723636) **4.** After remote destination answer, call is offered to A1 **5.** A1 answer **6.** App start recording on RD1 **7.** App stop recording on RD1 **8.** RD1 disconnects the call	Step 5- Check recording started on RD1. Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv. Check RD1 TermConnection.getCiscoRecorderInfo(): .getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT .getTerminalName() = central recorder device name .getAddress() = Recording profile (DN 2000) .getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW .getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 25

FI: Call Park- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording and A park and park reversion

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • recording profile:rec_profile1: 2000
• Call park number P1 on same cluster of A with park reversion number set to A1
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start recording on RD1
6. A1 park call to park number P1
7. Wait for park reversion happen
8. A1 answer the call
9. App stop recording on RD1
10. A1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Recording *not* retriggered on RD1.
	Step 7- Park Reversion happens. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 answers. Recording remains.
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 26

FI: Barge- devRD (SME), RDD (cluster2), A (leaf), A' (leaf) - RD selective recording and A' does Barge

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2010) configured as:
  • GW preferred
  • selective recording enabled
• devA' (SEP1396) on cluster 3 has line A1' (2010) configured as:
  • GW preferred
  • selective recording enabled
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. A1’ does barge
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- A1' barge in succeeds. Check barge events.
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER_INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 27

FI: Dpark- devRD (leaf), RDD (cluster2), A (SME), B (SME) - RD selective recording and A dpark and B retrive

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1621) (active remote destination: 1721711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• dpark DN: D1 (7001) is in same cluster of A, a prefix of 666 is to retrieve.
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. App start recording on RD1
6. A1 transfer call to dPark number D1
7. B1 calls dpark D1
8. Stop recording on RD1
9. B1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Recording *not* retriggered on RD1.
	Step 7- B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 28

FI: Monitoring- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD selective recording and B start monitoring

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP3B5F) on cluster 3 has line B1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB is the Supervisor
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 start monitoring on A1
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- Check A1 get CiscoTermConnMonitoringStartEv.
	A1: CiscoTermConnMonitorTargetInfoEv
	B1: CiscoTermConnMonitorTargetInfoEv
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv
	Check A1 get CiscoTermConnMonitoringEndEv.

Scenario 29

FI: Whisper coaching- devRD (leaf), RDD (cluster2), A (SME) and B (SME) - RD selective recording and B start whisper coaching

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SIP2) on cluster 1 has line A1 (9001) configured as:
  • no recording
• devB (SIP3) on cluster 1 has line B1 (9002) configured as:
  • no recording
• devB is the Supervisor
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. B1 start whisper coaching on A1
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- Check A1 get MonitoringStartEvent.
	A1:CallAttributeInfoEvent with type = coaching
	B1:CallAttributeInfoEvent with type = coaching
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 30

FI: Agent Greeting- devRD (SME), RDD (cluster2), A (leaf) and B (leaf) - RD selective recording and B start agent greeting

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (CTI1) on cluster 3 has line B1 (1600) configured as:
  • no recording
• devB is the IVR
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answer the call
5. Start agent greeting on A1 with B1 as IVR DN
6. App start recording on RD1
7. App stop recording on RD1
8. A1 disconnects the call
	Step 5- Check A1 get CiscoMediaStreamStartedEv (CTI: sendMediaToBIBStartEvent).
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 31

FI: Persistent connection- devRD (SME), RDD (cluster2), A (leaf) - RD auto recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. App makes CreatePersistentConnection() request on RD1
4. A1 call RD1
5. Remote device answer
6. RD1 disconnects the call
	Step 3- Persistent connection is created on RD1.
	Step 5- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 32

FI: Call pick up- devRD (SME), RDD (cluster2), A (leaf), B (leaf)- RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (CTI1) on cluster 3 has line A1 (1600) configured as:
  • no recording
• devB (SIP1) on cluster 3 has line B1 (9000) configured as:
  • no recording
• A1 and B1 are in pick up group: PG_1
• Service parameter: auto pick up enabled
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call B1
4. After remote destination answer, call is offered to B1 and A1
5. A1 pick up the call
6. App start recording on RD1
7. App stop recording on RD1
8. RD1 disconnects the call
	Step 5- After pickup, A1 and RD1 are connected.
	Step 6- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION _INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 33

FI: CTI Failover- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD (CTI Port) on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devRD in device pool DPPS1 (ccm1 (cluster1 PUB), ccm2 (cluster1 SUB))
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call A1
4. Remote destination answers, and then A1 answer
5. App start recording on RD1
6. Stop CTI Manager service on Pub1
7. Open provider on Sub1 (SME/Cluster1)
8. Reopen RD1
9. App start recording on RD1
10. RD1 disconnects the call
11. Start CTI Manager service on Pub1
	Step 4- A1 and RD1 are connected.
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Stop CTI Manager service on Pub1 successfully.
	Step 7- Open provider on Sub1 successfully.
	Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 34

FI: CPN- devRD (SME), RDD (cluster2), A (leaf) - RD selective recording

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD (CTI Port) on cluster 2.
• devA (SEPDB17) on cluster 3 has line A1 (2010) configured as:
  • Phone preferred
  • selective recording enabled
  • rec_profile1: 2000
• devRD in device pool DPPS1 (ccm1 (cluster1 PUB), ccm2 (cluster1 SUB))
• SIP trunk configured for CPN on SME/cluster1 and leaf, RD set up CPN also.
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. RD1 call A1
4. Remote destination answers, and then A1 answer
5. App start recording on RD1
6. Stop CTI Manager service on Pub1
7. Open provider on Sub1
8. Reopen RD1
9. App start recording on RD1
10. RD1 disconnects the call
11. Start CTI Manager service on Pub1
	Step 4- A1 and RD1 are connected.
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 6- Stop CTI Manager service on Pub1 successfully.
	Step 7- Open provider on Sub1 successfully.
	Step 8- Check RD1 get ExistingCallEvent and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_APPLICATION_INITIATED_SILENT
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 35

FI: Redirect- local- devRD (leaf), RDD (cluster2), A (leaf) and B (leaf) - RD auto recording and A redirect

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTI_RD3) on cluster 3 has line RD1 (1622) (active remote destination: 1721711681 on cluster2) configured as:
  • auto recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (SEP3B5F) on cluster 3 has line A1 (3601) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEPDB17) on cluster 3 has line B1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. A1 redirect to B1
6. B1 answer
7. RD1 disconnects the call
	Step 4- Check auto recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 5- A1 redirects.
	Step 6- B1 answers. B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 7- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 36

FI: Transfer- local-devRD (SME), RDD (cluster2), A (SME) and B (SME) - RD selective recording and A transfer

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. App start user recording on RD1
6. A1 setup transfer to B1
7. B1 answer
8. A1 complete transfer
9. Stop recording on RD1
10. RD1 disconnects the call
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete transfer. B1 and RD1 are connected. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 37

FI: Conference- local-devRD (SME), RDD (cluster2), A (SME) and B (SME) - RD selective recording and A conference

Setup and Action	Events and Call Info

Cluster 1 (SME):
• "Allow trunk use GW recording" is enabled
• Central recorder: 2000
Cluster 2:
Cluster 3 (leaf):
• "Allow trunk use GW recording" is enabled
• Branch recorder: 2000
• devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  • selective recording enabled
  • GW preferred
Remote Destination RDD on cluster 2.
• devA (IP10) on cluster 1 has line A1 (2303) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
• devB (SEP334F) on cluster 1 has line B1 (2205) configured as:
  • GW preferred
  • selective recording enabled
  • rec_profile1: 2000
1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, and B1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. App start user recording on RD1
6. A1 setup transfer to B1
7. B1 answer
8. A1 complete transfer
9. App stop recording on RD1
10. RD1 disconnects the call
	Step 5- Check recording started on RD1.
	Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 8- A1 complete conference. A1, B1 and RD1 are in conference. Recording retriggered on RD1.
	Check RD1 get CiscoTermConnRecordingEndedEv , CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.
	Check RD1 TermConnection.getCiscoRecorderInfo():
	.getRecordingType() = CALL_RECORDING_TYPE_USER _INITIATED_FROM_APPLICATION
	.getTerminalName() = central recorder device name
	.getAddress() = Recording profile (DN 2000)
	.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_ DEVICE_TYPE_GW
	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW
	.getProtocolReferenceGUID() =
	.getMediaForkingClusterID() = name of cluster1
	Step 9- Check RD1 get CiscoTermConnRecordingEndedEv

Recording Fail Event

Scenario 38

Failure Event: Hold Resume- auto recording on IP phone - devRD, A, C, D (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD1) on cluster 1 has line RD1 (2303) (active remote destination: 1711623 on cluster2) configured as:
  - selective recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2206) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, C1, and D1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers, and then A1 answer
5. A1 put call on hold
6. C1 call D1, D1 answer
7. A1 resume

Step 4- Check auto recording started on A1.

Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check A1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- A1 put call on hold. Check A1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 7- A1 resume. No CiscoTermConnRecordingFailedEv.

Step 8- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 10- Check auto recording started on A1.

Check A1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check A1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

8. C1 drop **9.** A1 put on hold **10.** A1 resume **11.** A1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 11- Check A1 get CiscoTermConnRecordingEndedEv

Scenario 39

Failure Event: Redirect- auto recording on CTI remote device - devRD, A, C, D, B, E (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  - auto recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  - GW preferred
  - selective recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)
- devB (SEPAB21) on cluster 1 has line B1 (9000)
- devE (IP9) on cluster 1 has line E1 (2302)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, B1, C1, D1, and E1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers
5. Drop call from recorder
6. C1 call D1, D1 answer

Step 4- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- Drop call from recorder. Check RD1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 7-. A1 redirect and B1 answer. No CiscoTermConnRecordingFailedEv.

Step 9- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 10- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

7. A1 redirect to B1 **8.** B1 answer **9.** C1 drop **10.** B1 redirect to E1 **11.** E1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 11- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 40

Failure Event: Transfer- auto recording on CTI remote device - devRD, A, C, D, B, E (SME) and RDD (cluster2)

Setup and Action	Events and Call Info

Cluster 1 (SME):

- "Allow trunk use GW recording" is enabled
- Central recorder: 2000

Cluster 2:

Cluster 3 (leaf):

- "Allow trunk use GW recording" is enabled
- Branch recorder: 2000
- devRD (CTIRD3) on cluster 1 has line RD1 (9008) (active remote destination: 1711681 on cluster2) configured as:
  - auto recording enabled
  - GW preferred

Remote Destination RDD on cluster 2.

- devA (SEP334F) on cluster 1 has line A1 (2205) configured as:
  - GW preferred
  - selective recording enabled
  - rec_profile1: 2000
- devC (IP10) on cluster 1 has line C1 (3300) configured as:
  - GW preferred
  - auto recording enabled
  - rec_profile1: 2000
- devD (SEP3925) on cluster 1 has line D1 (9002)
- devB (SEPAB21) on cluster 1 has line B1 (9000)
- devE (IP9) on cluster 1 has line E1 (2302)

1. Open provider Prov1, Prov2, Prov3
2. Observe RD1, RDD1, A1, B1, C1, D1, and E1 in Prov1/Prov2/Prov3 accordingly
3. A1 call RD1
4. Remote destination answers
5. Drop call from recorder
6. C1 call D1, D1 answer

Step 4- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

.getMediaForkingDeviceName() = SIP trunk device name - SIP GW

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 5- Drop call from recorder. Check RD1 get CiscoTermConnRecordingEndedEv

Step 6- Check auto recording started on C1.

Check C1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check C1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_PHONE

.getMediaForkingDeviceName() = Device name of C

.getProtocolReferenceGUID() =

.getMediaForkingClusterID() = name of cluster1

Step 9-. A1 complete transfer. No CiscoTermConnRecordingFailedEv.

Step 10- C1 drop. Check C1 get CiscoTermConnRecordingEndedEv

Step 13- Check auto recording started on RD1.

Check RD1 get CiscoTermConnRecordingStartedEv and CiscoTermConnRecordingTargetInfoEv.

Check RD1 TermConnection.getCiscoRecorderInfo():

.getRecordingType() = CALL_RECORDING_TYPE_AUTOMATIC

.getTerminalName() = central recorder device name

.getAddress() = Recording profile (DN 2000)

.getMediaForkingDeviceType() = CALL_RECORDING_MEDIA_FORKING_DEVICE_TYPE_GW

7. A1 setup transfer to B1 **8.** B1 answer **9.** A1 complete transfer **10.** C1 drop **11.** B1 transfer to E1 **12.** E1 answer **13.** B1 complete transfer **14.** E1 disconnect the call	.getMediaForkingDeviceName() = SIP trunk device name - SIP GW .getProtocolReferenceGUID() = .getMediaForkingClusterID() = name of cluster1 Step 14- Check RD1 get CiscoTermConnRecordingEndedEv

Scenario 41

Cluster ID in Open Provider: Open providers after changing cluster ID

Setup and Action	Events and Call Info
Cluster 1 (SME): • "Allow trunk use GW recording" is enabled • Central recorder: 2000 Cluster 2: Cluster 3 (leaf): • "Allow trunk use GW recording" is enabled • Branch recorder: 2000 Remote Destination RDD on cluster 2. **1.** Open provider on cluster 3 Leaf **2.** Change cluster ID on cluster 3 (e.g. “ClusterArcadia2013”) **3.** Restart CTI and CCM services **4.** Provider is reopened on cluster 3 Leaf **5.** Change cluster ID on cluster 3 back to original (e.g. “ClusterID3”) **6.** Restart CTI and CCM services **7.** Provider is reopened on cluster 3 Leaf **8.** Close provider	Step 4- After provider is reopened, check to see new cluster ID via VerifyProviderInfo on getClusterID. Step 4- After provider is reopened, check to see original cluster ID via VerifyProviderInfo on getClusterID.

Secured Recording

Secured Recording Use Cases

Scenario	Expected Result	Info
Scenario 1 1 .Agent and recorder are encrypted 2. Customer is in a secured/non-secured call with the agent. 3. The agent initiates a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv TermA CiscoTermConnRecordingTargetInfoEv	
Scenario 2 1. Agent is non-secured and recorder is encrypted 2. Customer is in a non-secured call with agent 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv TermA CiscoTermConnRecordingTargetInfoEv	
Scenario3 1. Agent is encrypted and recorder is non-secured 2. Customer is in a secured/non-secured call with the agent 3. The agent issues a request to record the call	CallCtlTermConnTalkingEv TermA CiscoTermConnRecordingStartEv CiscoTermConnRecordingEndEv	Cause = CAUSE_BCNAUTHORISED
Scenario 4 1. Agent and Recorder are non-secured 2. Customer and Agent are in a non-secured call 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request succeeds CiscoTermConnRecordingStartEv termA CiscoTermConnRecordingTargetInfoEv	
Scenario 5 1. Agent and recorder are authenticated 2. Agent and customer are in a call 3. Agent issues a request to record the call	CallCtlTermConnTalkingEv TermA Request fails with exception	

Monitoring and Recording Use Cases

Scenario	Expected Result
Scenario 1 1. Agent is non-secured, Supervisor and recorder are encrypted 2. Agent is in a non-secured call with customer 3. Supervisor monitors the call in non-secured mode	GC1: CallCtlTermConnTalkingEv TermA GC2: CallActiveEv GC2: ConnCreatedEv S …. …. GC2: CallCtlConnEstablishedEv TermS GC1: CiscoTermConnMonitorStartEv TermA GC1: CiscoTermConnMonitorInitiatorInfoEv TermA GC2: ConnCreatedEv A …. …. GC2: CallCtlConnEstablishedEv TermA GC2: CiscoTermConnMonitorTargetInfoEv TermS
4. Application requests for recording the call at Supervisor	CiscoTermConnRecordingStartEv CallCtlTermConnTalkingEv TermS The recording is done in a secured mode as Both the supervisor and recorder are secured, but the overall call security is non-secured.

Redirect Set OriginalCalledID

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

Scenario One

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

Application will see following events for parties A and B:

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

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

Scenario Two

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

The application will see following events for party D:

.

Meta Event Cause	Call	Event	Fields
META_CALL_STARTING	Call1	CallActiveEv ConnCreatedEv for D ConnInProgressEv for D CallCtlConnOfferedEv for D ConnCreatedEv for A CallCtlConnInitiatedEv for A	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D
META_CALL_PROGRESS	Call1	ConnAlertingEv for D CallCtlConnAlertingEv for D TermConnCreatedEv for D CallCtlTermConnRingingEv for D ConnConnectedEv for A CallCtlConnEstablishedEv for A	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D
META_CALL_PROGRESS	Call	ConnConnectedEv for D CallCtlConnEstablishedEv for D TermConnActiveEv for D CallCtlTermConnTalkingEv for D	CallingParty = A CalledParty = D **LastRedirectedParty = C** CurrentCalledParty = D

Redirect to a Device

The following tables display message sequence charts for the Redirect enhancement that allows you to redirect calls to a specific device, even if that device is sharing a line with another device.

CallRedirect to Shared Line with Device Name

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls B and B answers	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B	

Action	Events	CallInfo
Application redirects the call from B to C using CiscoConnection.redirect() method with devcieName as C	GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 CallCtlConnOfferedEv C GC1 ConnAlertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEvImpl TermC GC1 TermConnCreatedEv TermC' GC1 TermConnPassiveEv TermC' GC1 CallCtlTermConnInUseEv TermC' GC1 TerConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT

CallRedirect to Shared Line with Invalid Device Name

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls B and B answers	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAltertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEv TermB GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B	
Application redirects the call from B to C using CiscoConnection.redirect() method with deviceName as D		InvalidPartyException caught: geterrorcode() = CiscoJtapiException. REDIRECT_CALL_INVALID _DEVICE_NAME

CallRedirect to Shared Line using selectRoute

In this use case devices A, B, C, and C' are IP phones where C and C' share a line. RP is the route point.

Action	Events	CallInfo
A calls route point RP. RP redirects the call to the destination C using selectRoute() method	GC1 CallActiveEv A GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEV TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv RP GC1 ConnInProgressEv RP GC1 CallCtlConnOfferedEv RP	
RP redirects the call to the destination C using selectRoute() method	GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 ConnAltertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEv TermC GC1 ConnCreatedEv C' GC1 TermConnPassiveEv TermC' GC1 CallCtlTermConnInUseEv TermC'	CurrentCallingParty = A CurrentCalledParty = C Reason = CiscoFeatureReason.REASON_REDIRECT

Verify Remote Destination Support

Table 83: Verify Remote Destination in Add Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("testRDD", "77000", true)	CiscoProvTerminalRemoteDestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestinationInfo[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationName() = "testRDD" CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "77000" CiscoRemoteDestination Info[0].getIsActiveRD() = true

Table 84: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("77000", "testRDD", "79000", true)	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestinationInfo[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationName() = "testRDD" CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "79000" CiscoRemoteDestination Info[0].getIsActiveRD() = true

Table 85: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Action	Events	Call Info
User1 invokes CiscoRemoteTerminal. updateRemoteDestination Number("77000", "79000")	CiscoProvTerminalRemote DestinationChangedEv	CiscoProvTerminalRemote DestinationChangedEv. getRemoteDestinations() which returns an array of all remote destinations configured for that remote terminal, CiscoRemoteDestination Info[0] (assuming only one configured) CiscoRemoteDestination Info[0].getRemoteDestinationNumber() = "79000"

Table 86: Verify Remote Destination in Add Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. addRemoteDestination ("testRDD", "99000", true)	Caught exception com.cisco.jtapi.PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_ CONNECT_DESTINATION _NOT_REACHABLE.

Table 87: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestination()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestination ("77000", "testRDD", "99000", true)	Caught exception com.cisco.jtapi. PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_CONNECT _DESTINATION_NOT_REACHABLE.

Table 88: Verify Remote Destination in Update Where Route Pattern Configured Is 7.XXXX and Destination Is Not Reachable. User1 Has cti Remote Device in Its Control List and Existing Remote Destination of 77000 Configured. User Invokes CiscoRemoteTerminal.updateRemoteDestinationNumber()

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. updateRemoteDestinationNumber ("77000", "99000")	Caught exception com.cisco.jtapi. PlatformExceptionImpl: Extend and Connect destination is not reachable	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex). getErrorCode() = CiscoJtapiException. CTIERR_EXTEND_AND_CONNECT _DESTINATION_NOT_REACHABLE.

Secure Conferencing

Action	Events	Call info
Scenario:1 Configuration: A (secure) and B (secure). A calls B. B answers. Application issues getCallSecurtyStatus().	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 Returns the call security status of the call.	Calling: A Called: B CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info. CallSecurityStatus = 3 (ENCRYPTED).
Configuration: A (secure), B (secure) and C (non-secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv () A calls B. B answers. B call C. C answers. B completes conference.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1.	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED). Note: Though CallSecurityStatus = NotAuthenticated, A and B will continue to have secure media between themselves and the conference bridge, i.e. they will continue to receive SRTP key info because they are encrypted parties.

Action	Events	Call info
Scenario:3 Configuration: A (secure), B (secure) and C (secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv () A calls B. B answers. B call C. C answers. B completes conference. Application issues getCallSecurtyStatus().	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1. Returns the call security status of the call (Secure).	Participants: A, B, C CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info.
Scenario:4 Application does not add call observers on A, B, C. Configuration: A (secure), B (secure) and C (non-secure). A calls B. B answers. B call C. C answers. B completes conference. Application later adds call observers on A, B, C. Application issues getCallSecurtyStatus().	CallSecurityStatusChangedEv for callID = GC1 with Cause = CAUSE_SNAPSHOT Returns the call security status of the call.	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info.
Scenario:5 Configuration: A (secure), B (secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A calls B. B answers. B puts call on hold. B resumes call.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 CallSecurityStatusChangedEv for callID = GC1 CallSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info. CallSecurityStatus = 0 (UNKNOWN) will get updated in the call info. CallSecurityStatus = (ENCRYPTED) will get updated in the call info.

Action	Events	Call info
Scenario:6 Configuration: A (secure), B (secure) and C (non-secure). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A calls B. B answers. B consults C. C answers. B completes transfer.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1 CallActiveEv for callID = GC2 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC2 CallCtlSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info for GC1. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC2. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC1.

Action	Events	Call info
Scenario:7 Configuration: A (secure), B (secure), C (secure), D (secure), and E (Authenticated). Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv() A, B and C are part of a conference Call 1. C, D and E are a part of another conference Call 2. C chains the 2 conferences.	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for C' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for D' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for E' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallCtlSecurityStatusChangedEv for callID = GC1	CallSecurityStatus = 3 (ENCRYPTED) will get updated in the call info for GC1. CallSecurityStatus = 2 (AUTHENTICATED) will get updated in the call info for GC2. CallSecurityStatus = 1 (NOTAUTHENTICATED) will get updated in the call info for GC1 and GC2.
Scenario:8 Configuration: A (secure), B (secure), B (authenticated) Application sets ini parameter = true by issuing enableSecurtyStatusChangedEv()	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID = GC1.	CallSecurityStatus = 2 (AUTHENTICATED) will get updated in the call info for GC1. **Note** Applications who have added call observers on B' will also get the event, i.e. the new event will be delivered to RIU Parties as well.

Secure Connection Enhancements

Action	Events	Call information
The application wants to connect securely to Cisco Unified Communications Manager and downloads the certficate using the interfaces in CiscoJtapiProperties. After down loading the certificate, application initializes provider using a providerString formatted with new parameters: String providerString = serverName + ";login = " + login + ";passwd = " + passwd + ";InstanceID = " + instanceID + ";CAPF = " + CAPFserver + ";CAPFPort = " + capfport + ";TFTP = " + TFTPServer + ";TFTPPort = " + tftpport + ";CertPath = " + certificatepath+ ";CertStorePassphrase = " + cartificatepassphrase +";" JtapiPeer peer = JtapiPeerFactory.getJtapiPeer (null); MyProviderObserver providerObserver = new MyProviderObserver (); provider = peer.getProvider (providerString);	Provider is initialized correctly through TLS connection. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals ProvInService event is delivered to provider observer	
The application is not interested in secure connection and open provider using userid and passwd in provider String String providerString = serverName + ";login = " + login + ";passwd = " + ";"	Provider is initialized correctly. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals ProvInService is delivered to provider observer	
The application uses jtprefs to download the certificates and initializes provider specifying only the userid, passwd, instanceid and certificate pass phrase in provider string. String providerString = serverName + ";login = " + login + ";passwd = " + passwd + ";InstanceID = " + instanceID +":CertStorePassphrase = " + cartificatepassphrase +";"	Provider is initialized correctly through TLS connection. Provider.getAddress() and provider.getTerminals() return correct number of addresses and terminals	

Secure Icon Enhancements

Enable the callSecurityStatusChangedEv using JTAPI ini parameters or using the JTAPIProperties.

Cluster1 and Cluster2 are secured and User is also a secured user having a CAPF profile associated with it. Enable "SRTP Allowed" in the SIP trunk Configurations.

TermA is registered to Cluster1 with address A.

TermB is registered to Cluster2 with address B

TermC is registered to Cluster2 or Cluster1 as per Use case and has address C

SIP trunk is configured on Cluster1 to make calls to cluster2

A Route Pattern is configured on Cluster 1 to route the call to the SIP trunk

Use Case One

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1. Security Mode of Both term A and termB is encrypted. Security mode of the SIP trunk is non secured.		
A calls B though the route pattern.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	
Call is offered on B and B accepts.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Two

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1. Security Mode for term A and termB and term C is encrypted. Security mode of the SIP trunk is non secured.		
A calls B though the route pattern.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	

Action	Expected results	Information
Call is offered to B and B accepts.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B answers the call.	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED
B Redirects the call to C	TermA CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermA CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 ConnCreatedEv C GC1 ConnInProgressEv C GC1 CallCtlConnOfferedEv C GC1 ConnAlertingEv C GC1 CallCtlConnAlertingEv C GC1 TermConnCreatedEv TermC GC1 TermConnRingingEv TermC GC1 CallCtlTermConnRingingEvImpl termC GC1 TermConnDroppedEv TermB GC1 CallCtlTermConnDroppedEv TernB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B	call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
C answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Three

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When Using only sRTP' in the SIP trunk config on cluster1 Security Mode of Both term A and termB and term C is encrypted Security mode of the SIP trunk is non secured		
A calls B though the route pattern	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	

Action	Expected results	Information
Call is offered on B and B accepts	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED
B calls C	GC1 CallCtlTermConnHeldEv TermB GC1 CiscoCallSecurityStatusChangedEv GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConenctedEvB GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TermB GC2 TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 CallCtlConnDialingEv B GC2 ConnEstablishedEv B	Ev.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
Call is offered on C and C accepts	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv TermC GC2 TermConnRingingEv TermC GC2 CallCtlTermConnRingingEvImpl termC	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
C answers the call	TermB CiscoRTPOutputStartedEv GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv TermC TermB CiscoRTPInputStartedEv TermC CiscoRTPOutputStartedEv TermC CiscoRTPInputStartedEv GC2 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Action	Expected results	Information
B does a Direct Transfer GC1.transfer(GC2)	GC1 CiscoTermConnSelectChangedEv termB GC2 CiscoTermConnSelectChangedEv TermB TermC CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermC CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnEstablishedEv C GC1 TermConnCreatedEv TermC Gc1 TermConnActiveEv TermC Gc1 CallCtlTermConnTalkingEv TermC GC2 TermConnDroppedEv TermC GC2 CallCtlTermConnDroppedEv TermC GC2 ConnDisconnectedEv C GC2 CallCtlConnDisconnectedEv C GC2 TermConnDroppedEv termB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectedEv B GC2 CallInvalidEv GC1 TermConnDroppedEv termB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CiscoTransferEndEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv GC1 CiscoCallSecurityStatusChangedEv	Ev.getCallSecurityStatus() = CALLSECURITY_ENCRYPTED

Use Case Four

Action	Expected results	Information
Set the value for "Consider Traffic on this Trunk Secure" to 'When using both sRTP and TLS' in the SIP trunk config on cluster1 Security Mode of Both term A and termB and term C is encrypted Security mode of the SIP trunk is non secured		
A calls B though the route pattern	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConenctedEvA GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEv TermA GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 ConnEstablishedEv A	
Call is offered on B and B accepts	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TermB GC1 TermConnRingingEv TermB GC1 CallCtlTermConnRingingEvImpl termB	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B answers the call	TermA CiscoRTPOutputStartedEv GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv TermB TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
B calls C	GC1 CallCtlTermConnHeldEv TermB GC2 CallActiveEv GC2 ConnCreatedEv B GC2 ConnConenctedEvB GC2 CallCtlConnInitiatedEv B GC2 TermConnCreatedEv TermB GC2 TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 CallCtlConnDialingEv B GC2 ConnEstablishedEv B	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED
Calls is offered on C and C accepts	GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAlertingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv TermC GC2 TermConnRingingEv TermC GC2 CallCtlTermConnRingingEvImpl termC	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

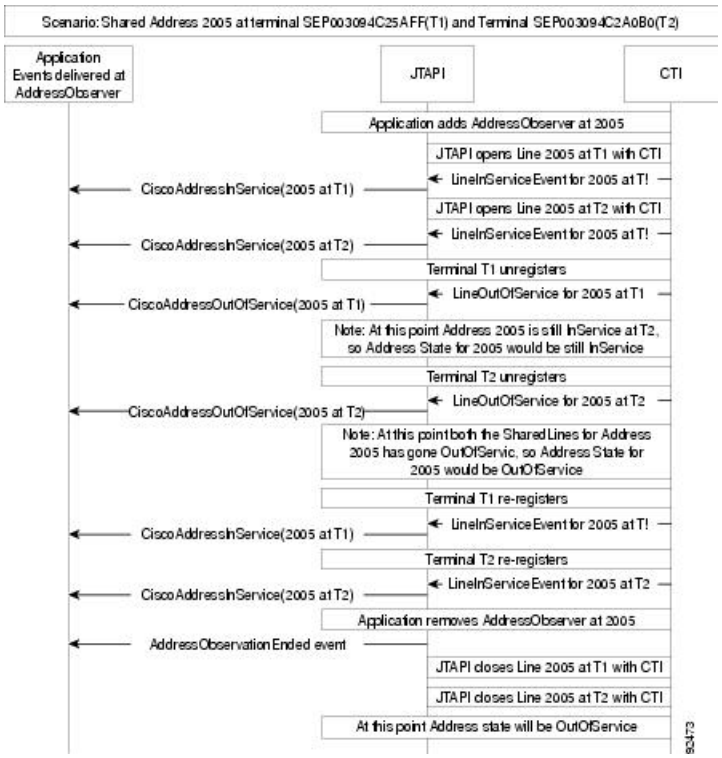
Action	Expected results	Information
C answers the call	TermB CiscoRTPOutputStartedEv GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv TermC TermB CiscoRTPInputStartedEv TermC CiscoRTPOutputStartedEv TermC CiscoRTPInputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

Action	Expected results	Information
B does a Direct Transfer GC1.transfer(GC2)	GC1 CiscoTermConnSelectChangedEv termB GC2 CiscoTermConnSelectChangedEv TermB TermC CiscoRTPOutputStoppedEv TermB CiscoRTPOutputStoppedEv TermC CiscoRTPInputStoppedEv TermB CiscoRTPInputStoppedEv GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1 ConnConnectedEv C GC1 CallCtlConnEstablishedEv C GC1 TermConnCreatedEv TermC Gc1 TermConnActiveEv TermC Gc1 CallCtlTermConnTalkingEv TermC GC2 TermConnDroppedEv TermC GC2 CallCtlTermConnDroppedEv TermC GC2 ConnDisconnectedEv C GC2 CallCtlConnDisconnectedEv C GC2 TermConnDroppedEv termB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectedEv B GC2 CallInvalidEv GC1 TermConnDroppedEv termB GC1 CallCtlTermConnDroppedEv TermB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CiscoTransferEndEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermB CiscoRTPInputStartedEv TermA CiscoRTPOutputStartedEv	Call.getCallSecurityStatus() = CALLSECURITY_NOTAUTHENTICATED

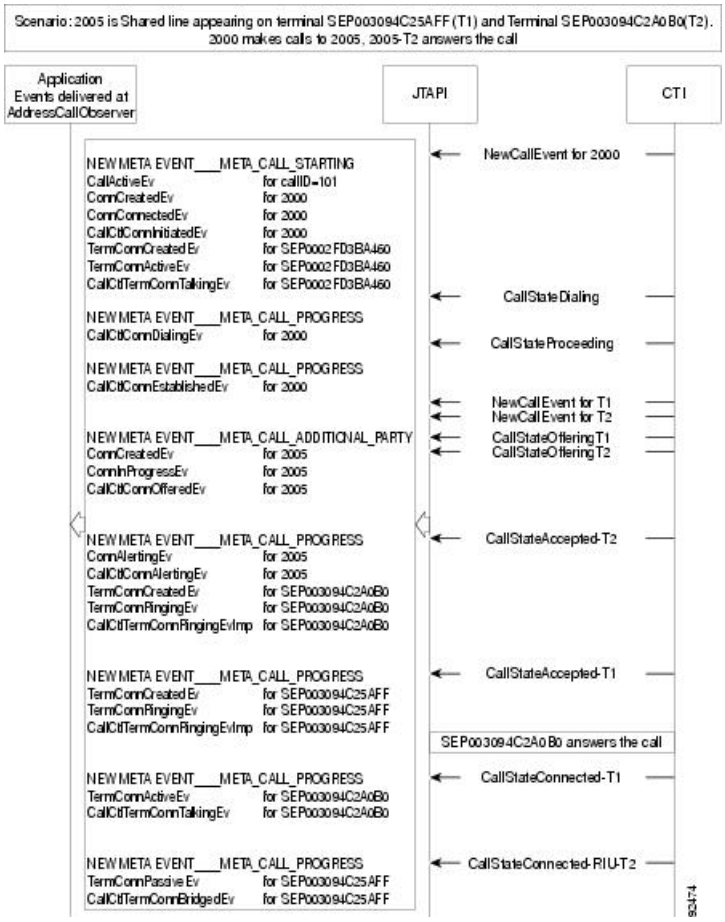
Shared Line Support

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

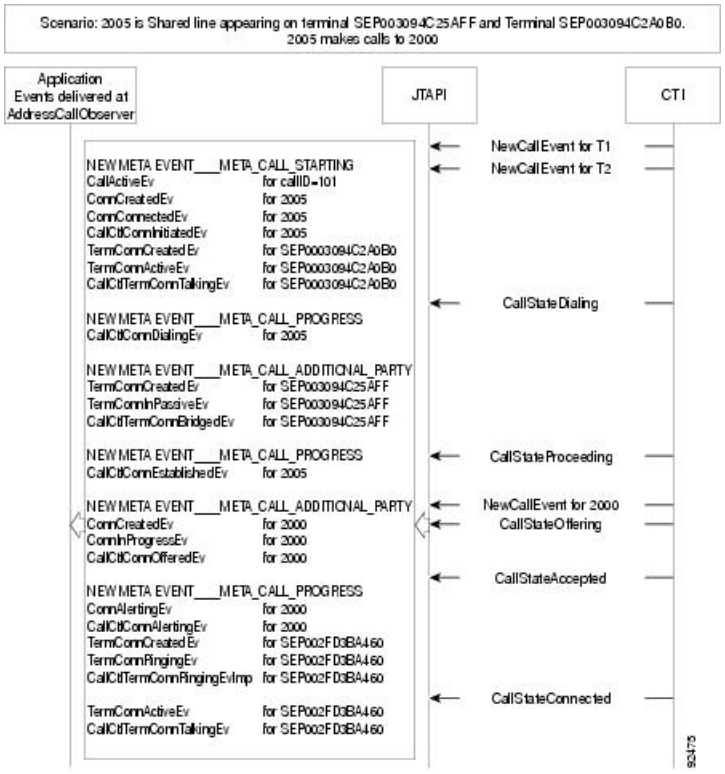
AddressInService/AddressOutofService Events



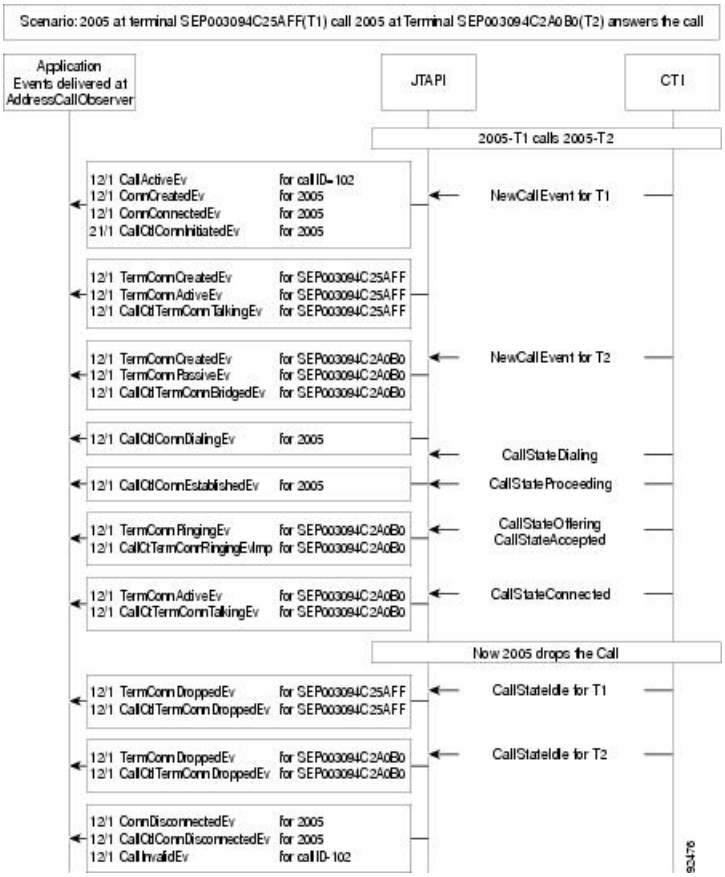
Incoming Call to Shared Address



Outgoing Call From Shared Address



Shared Address Calling Itself



Single Sign-On

Here are the list of use cases for this feature.

Sl. No	Scenario	Result
1.	Start the application (cucimoc) and getProvider(str) API is called by application specifying the singlesignon ticket. Application calls getAddresses() API.	Returns provider to the application and all the addresses configured in the control list.
2.	Application specifies an invalid ticket but correct userid and password in API.	Throws a platform exception to getProvider() API.

Sl. No	Scenario	Result
3.	Start the application and it calls the getProvider() API with singlesignon ticket. The network connectivity is lost. Application gets a new singlesignon token and calls getProvider().	Returns provider to the application. Delivers ProvOutOfServiceEv to provider observer. JTAPI connects and tries to authenticate the user which fails. Delivers ProvShutdownEv to provider observer. Returns provider object to the application.
4.	Start the application and getProvider() API is called by application with the singlesignon ticket. But the feature is not enabled on Cisco Unified Communications Manager.	Throws PlatformException and getErrorCode() returns CiscoJTAPIException - CTIERR_SSO_DISABLED.
5.	Start the application and getProvider() API is called by application with invalid singlesignon ticket.	Throws PlatformException and getErrorCode() returns CiscoJTAPIException - CTIERR_DIRECTORY_LOGIN_FAILED.
6.	Multiple providers: Start the application and getProvider() on two nodes in the cluster with the same token.	Both getProvider() call is successful with the first provider. Throws exception to the second getProvider().

Single Step Transfer

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

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

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

Action	Address A (5001) Terminal CTIP1	Address B (5002) Terminal CTIP2	Address C (5003) Terminal CTIP3
Call.transfer(string) (continued)			TermConnActiveEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStoppedEv Cause: CAUSE_NORMAL CiscoRTPOutputStoppedEv Cause: CAUSE_NORMAL ConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL ConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL META_UNKNOWN CallInvalidEv [#32] Cause: CAUSE_NORMAL

SIP REPLACE

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

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

SN	Scenario	Events at A	Events at B	Events at C
2.	REPLACE with INVITE an early Dialog: A (Dailog1) is in Call with B (Dialog2) (GC1), B is ringing. C sends INVITE with REPLACE Dialog2 (GC2). After replace completed, A (Dialog1) and C (Dialog3) in the Call	GCID and CPIC with reason REPLACES, Cgpn = C, Cdpn = A, Ocdpn = A, Lrp = B JTAPI Events CiscoCallChangedEv - (GC1 -GC2) ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B–GC1 ConnDisconnectedEv –A -GC1 CallCtlConnDisconnectedEv -A–GC1 CallInvalid -GC1 CallActive -CG2 ConnCreatedEv –C -GC2 ConnConnectedEv –C -GC2 CallCtlConnEstablishedEv -C -CG2 ConnCreatedEv –A –GC2 ConnConnectedEv –A -GC2 CallCtlConnEstablishedEv -A -CG2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B	CSCE -Idle with reason REPLACES JTAPI Events: ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv -A -GC1 ConnDisconnectedEv –B -GC1 CallCtlConnDisconnectedEv -B -GC1 CallInvalidEv -GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES	NewCall/CSCE -Dialing/CSCE -Connected, with Cgpn = C, Ccdpn = A, Ocdpn = A, Lrp = B JTAPI Events: CallActiveEv –GC2 ConnCreatedEv –C –GC2 ConnConnectedEv -C -–GC2 CallCtlConnEstablishedEv -C–GC2 ConnCreatedEv –A–GC2 ConnConnectedEv A -–GC2 CallCtlConnEstablishedEv -A -–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_REPLACES JTAPI CallInfo: Calling = C, Called = A, CurrentCalling = C, CurrentCalled = A, LastRedirecting = B

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

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

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

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

SIP REFER

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

IN-Dialog REFER Scenario

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

Scenario One

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

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

JTAPI moves A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection into the "UNKNOWN" state.

CAUSE_CODE provided will be CAUSE_NORMAL, new API provides REASON_REFER.

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

CallInfo at B and C would be as follows:

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

JTAPI Application observing B will see:

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

JTAPI Application observing C will see:

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

Scenario Two

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

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

JTAPI will Disconnect/Drop A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API would provide REASON_REFER.

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

CallInfo at B and C will be as follows

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

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Three

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

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

JTAPI will Disconnect/Drop the Connection/CallControlConnection/TerminalConnection/

CallControlTerminalConnection for C. CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER and move A's Connection/CallControlConnection/

TerminalConnection/CallControlTerminalConnection from the "Unknown" state to the Connected/

Established/Active/Talking state.

CallInfo at A and B will be as follows

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

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

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

Scenario Four

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

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

JTAPI will create Connection/CallControlConnection/TerminalConnection/

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

CallInfo at B and C will be as follows:

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

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

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Five

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

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

JTAPI will create Connection/CallControlConnection for A again and drops Connection/

CallControlConnection/TerminalConnection/CallControlTerminalConnection for C.

CAUSE_CODE provided will be CAUSE_NORMAL and new API will provide REASON_REFER.

CallInfo at A and B will be as follows

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

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

JTAPI Application observing A will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

JTAPI Application observing C will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = A

getCurrentCalledParty() = B

getLastRedirecting() = NULL

Scenario Six

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

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

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

CallInfo at B and C will be as follows

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

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

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

Scenario Seven

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

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

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

CallInfo at B and D will be as follows

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

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

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

Scenario Eight

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

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

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

Callinfo when Call is offered at C:

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

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

CallInfo in final Call:

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

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

JTAPI Application observing B will see in final Call:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = B

getCurrentCalledParty() = D

getLastRedirecting() = C

Scenario Nine

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

B consult transfer to D, A(Referrer) REFERs B(Referee) to C(Refer to target), C is ringing,

Bcompletes the transfer. Attempt to transfer will fail while C is ringing.

Scenario Ten

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

B consult transfer to D, A(Referrer) REFERs B(Referee) to C(Refer to target), C answers the call.

Refer will be successful. B completes the transfer, transfer will be successful, C and D will be in call.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API will provide REASON_REFER.

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

CallInfo at D and C would be as follows

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

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

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = C

getCurrentCalledParty() = D

getLastRedirecting() = B

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = C

getCurrentCalledParty() = D

getLastRedirecting() = B

Scenario Eleven

B is in a call with D, B consults to A(SIP UA in cluster/in control).

A(Referrer) REFERs B(Referee) to C(Refer to target), C is ringing, B completes the transfer.

REFER would fail. Call at A will be dropped, transfer is successful, D is getting RingBack, C is ringing.

JTAPI Disconnect/Drops A's Connect/CallControlConnection/TerminalConnection/

CallControlTerminalConnection. CAUSE_CODE provided will be CAUSE_NORMAL and the new API would provide REASON_REFER, Application will not know if REFER failed.

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

CallInfo at D and C would be as follows:

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

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

JTAPI Application observing D will see:

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty() = D

getCurrentCalledParty() = C

getLastRedirecting() = B

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty() = D

getCurrentCalledParty() = C

getLastRedirecting() = B

OutOfDialog Refer

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

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

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

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

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

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

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

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

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

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

JTAPI Application observing B will see:

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

JTAPI Application observing C will see:

getCallingParty() = B

getCalledParty() = C

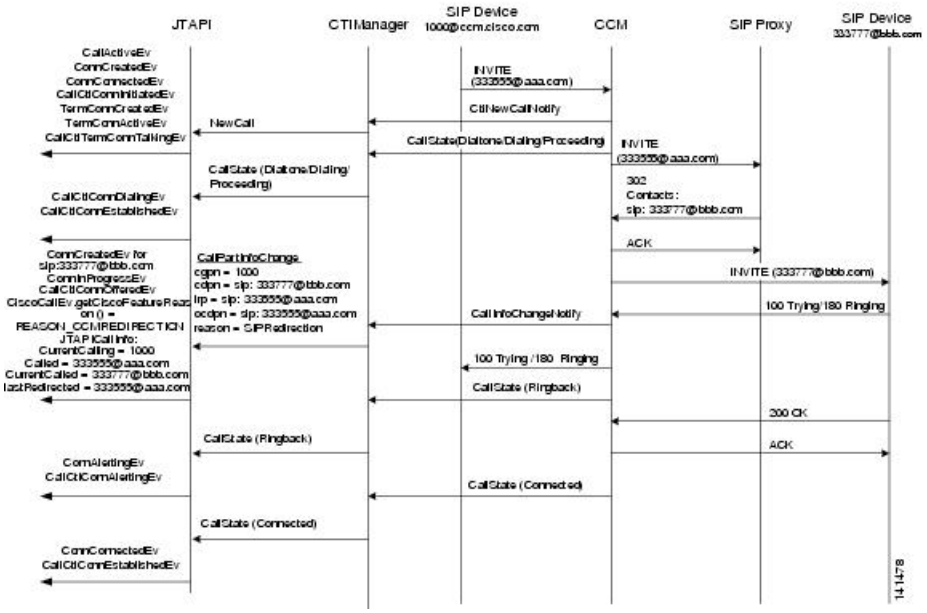
getCurrentCallingParty() = B

getCurrentCalledParty() = C

getLastRedirecting() = A

SIP 3XX Redirection

3XX Redirection – 302 Moved Temporarily



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

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

JTAPI reports CallActiveEv and Connection and CallCtlConnection events for 1000

JTAPI reports CallCtlConnEstablishedEv

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

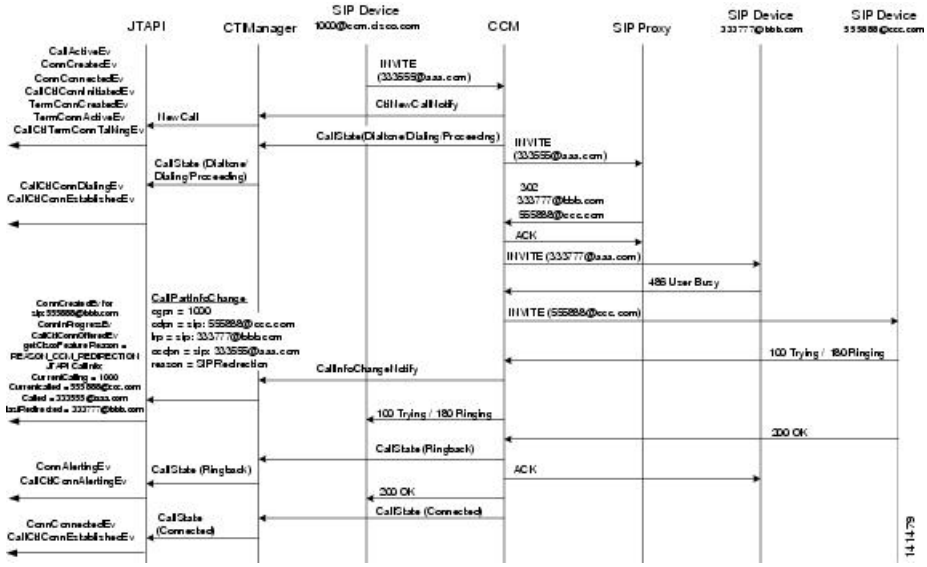
CallPartyInfoChange event is reported to application based on the SIPAlertInd from a Cisco Unified Communications Manager, if the called party information is changed.

JTAPI reports connection created events for 333777@bbb.com

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

JTAPI reports ConnAlertingEv and ConnEstablishedEv for far end.

3XX Redirection – Contact Busy



JTAPI CTI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

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

JTAPI reports CallActiveEv and Connection and CallCtlConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv

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

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

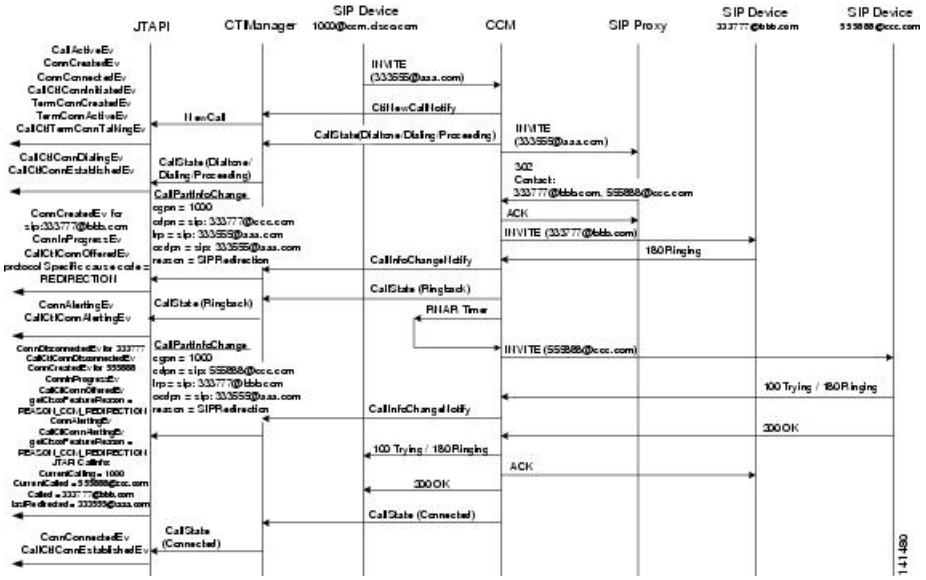
CallPartyInfoChange event is reported to application based on the SIPAlertInd from the Cisco Unified Communications Manager if the called party information is changed.

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

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

JTAPI reports CallCtlConnAlertingEv and CallCtlConnEstablishedEv for the new party

3XX Redirection – Contact Does Not Answer



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

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

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

SIP proxy reports a 302 for 333555@aaa.com. Based on the 302 the Cisco Unified Communications Manager initiates a call to the first contact in the Target list based on the q value to 333777@bbb.com. The Cisco Unified Communications Manager starts the RNAR timer.

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

JTAPI reports connection created events for 333777

RNAR timer expires and based on this expiration the Cisco Unified Communications Manager initiates a call to 555888@cisco.com.

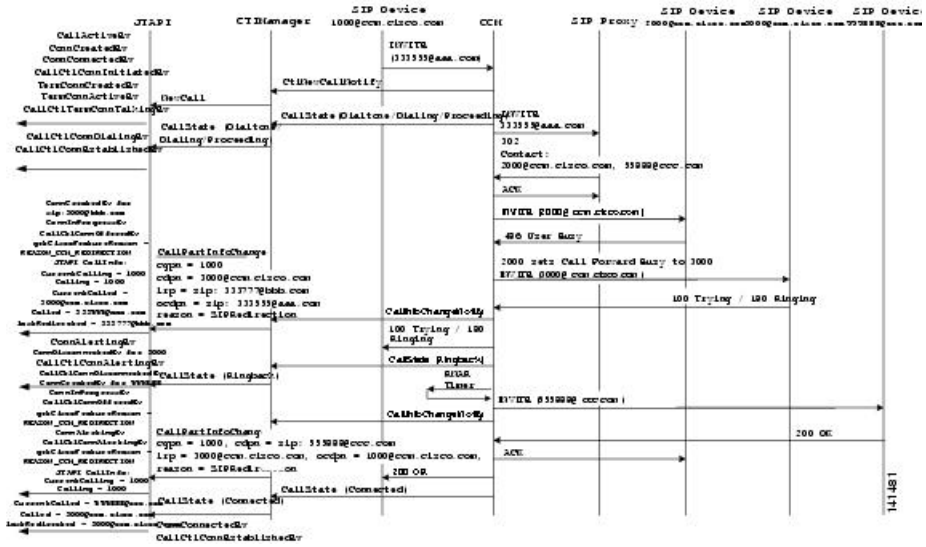
CallPartyInfoChange event is reported to application based on the SIPAlertInd/CcNotifyReq from the Cisco Unified Communications Manager if the called party information is changed.

JTAPI removes connection for 333777 and creates connection for 555888

CTI also reports CtiCallStateNotify (Connected).

JTAPI reports CallCtlConnEstablishedEv for 555888

3XX Redirection – Contact Within Cisco Unified Communications Manager Cluster Configured with Call Forward



JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

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

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

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

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

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

JTAPI reports connection created event for 3000

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

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

JTAPI destroys connection for 3000 and creates connection for 555888

CTI also reports CtiCallStateNotify (Connected).

JTAPI reports CallCtlConnEstablishedEv for 555888

3XX Redirection – Non-Available Target Member

JTAPI application monitors 1000@ccm.cisco.com

Cisco Unified Communications Manager user1000 initiates a call to 333555@aaa.com

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

JTAPI reports CallActiveEv and connection and terminalConnection events for 1000

CTI reports CtiCallStateNotify (Proceeding)

JTAPI reports CallCtlConnEstablishedEv for 1000

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

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

JTAPI reports connection created event for 2000

CTI also reports CtiCallStateNotify (Ringback/Connected).

JTAPI reports CallCtlConnAlertingEv and CallCtlConnEstablishedEv for 2000.

SIP Support

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

SIP Trunk Early Offer

Scenario One

Early offer call on a IPV4 mode. CTIPort or RP supports this feature.Application opens provider and adds address, terminal and call observers. (Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to the other address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT

Action	Result	Call information
Application sets the RTP parameters(IPv4 address and port) Application answers the call on B.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false

Scenario Two

Early offer call on a IPv4 mode. CTIPort or RP supports this feature. Application does not set RTP parameters in time(Fail Call Over SIP Trunk if MTP Allocation Fails = true).

Application opens provider and adds Address, terminal and call observers.(Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTermina.register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT for registration type and CiscoTerminal.IP_ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application does not sets the RTP parameters (IPv4 address and port).	GC1 TermConnDroppedEv TermA GC1 CallCtlTermConnDroppedEv TermA Gc1 ConnFailedEv A Gc1 CallCtlConnFailedEv A Gc1 CallInvalidEv PlatformException: Could not meet post condition of connect()	ev.getCause() = CAUSE_RESOURCES_NOT_AVAILABLE

Scenario Three

Early offer call on a IPv4 mode. CTIPort or RP supports this feature. Application does not setPort in time. (Fail Call Over SIP Trunk if MTP Allocation Fails = false)

Application opens provider and adds address, terminal and call observers. (Device = TermA address = A)

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTermina.register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application does not sets the RTP parameters. (IPv4 address and port) B answers the call. Application sets the RTP parameters.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv	isRTPRequired() = true.

Scenario Four

Early offer call on a dynamically registered IPv6 only CtiPort/RP with DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv6 for activeAddressingMode	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B	
App answers the call on B Application sets RTP parameters.	GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false

Scenario Five

Early Offer call on a dynamically registered CtiPort or RP with DYNAMIC_MEDIA_REGISTRATION for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new BaseMediaTerminal.register() API and DYNAMIC_MEDIA_REGISTRATION for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B.	
App answers the call on B Application sets RTP parameters	GC1 CiscoMediaOpenLogicalChannelEv TermA GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = true

Scenario Six

Two applications registering same CTIPort or RP with different values for registrationType.(dynamic).

Action	Result	Call information
Application1 registers the terminal TermA dynamically using the new CiscoBaseMediaTerminal .register() API and DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application2 registers the terminal TermA dynamically usig the new CiscoBaseMediaTerminal .register() API and passes something other than DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	PlatformException:	CTIERR_MEDIA_ALREADY_ TERMINATED_DYNAMIC_ GETPORT_SUPPORT
Application3 registers the terminal TermA dynamically usig the new CiscoBaseMediaTerminal .register() API and passes DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Scenario Seven

Application sets RTP parameters again for an early offer call with dynamically registered terminal having DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.

Action	Result	Call information
Application registers the terminal dynamically using the new CiscoBaseMediaTerminal .register() API DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

Action	Result	Call information
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application sets the RTP parameters (IPv4 address and port). Application answers the call on B. Application sets the RTP parameters again.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 CiscoMediaOpenLogicalChannelEv TermA InvalidStateException GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	ev.isRTPRequired() = false. CTIERR_OPERATION_NOT_ AVAILABLE_IN_CURRENT_STATE.

Scenario Eight

Transfer involving a early offer call

Application registers two terminals TermA and TermB dynamically with DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type and for both. A call is established between TermA and TermB.

Action	Result	Call information
A puts the call on hold	GC1 CallCtlTermConnHeldEv TermA	
A initiates a call to C	GC2 CallActiveEv GC2 ConnCreatedEv A GC2 ConnConnectedEv A GC2 CallCtlConnInitiatedEv A GC2 TermConnCreatedEv TermA GC2 TermConnActiveEvTerm A GC2 CallCtlTermConnTalkingEv TermA GC2 CallCtlConnDialingEv A GC2 CallCtlConnEstablishedEv A GC2 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes DYNAMIC_MEDIA_REGISTRATION_FOR GET_PORT_SUPPORT
Application sets the RTP parameters for TermA and opens the port(IPv4 address and port) App answers the call on C	GC2 TermConnRingingEv C GC2 CallCtlTermConnRingingEvImpl C GC2 CiscoMediaOpenLogicalChannelEv TermA GC2 ConnConnectedEv C GC2 CallCtlConnEstablishedEv C GC2 TermConnActiveEv C GC2 CallCtlTermConnTalkingEv C CiscoRTPInputStartedEvs CiscoRTPOutputStartedEvs GC2 ConnCreatedEv C GC2 ConnInProgressEv C GC2 CallCtlConnOfferedEv C GC2 ConnAletingEv C GC2 CallCtlConnAlertingEv C GC2 TermConnCreatedEv	Ev.isRTPRequired() = false

Action	Result	Call information
A tansfers the two calls GC1.transfer(GC2)	GC2 ConnDisconnectedEv C Gc2 CallCtlConnDisconnectedEv C GC2 ConnDisconnectedEv A GC2 CallCtlConDisconnectedEv A GC2 CallInvalidEv GC1 ConnCreatedEv C GC1 CiscoMediaOpenLogicalChannelEv TermB	Ev.isRTPRequired() = true
Application sets the RTP parameters for TermB	GC1 ConnEstablishedEv C CiscoRTPInputStartedEv CiscoRTPOutputStartedEvs	

Scenario Nine

Hold Resume Scenario

The application registers terminal TermA with DYNAMIC_MEDIA_REGISTRATION_FOR _GET_PORT_SUPPORT for registration type.

A call is established between TermA and TermB.

Action	Result	Call information
TermA puts the call on hold.	Gc1 CallCtlTermConnHeldEv CiscoRTPInputStopped EvCiscoRTPOutoutStoppedEv	
TermA resumes the call.	GC1 CiscoMediaOpenLogicalChannelEv TermA	Ev.isRTPRequired() = true
Application sets the RTP parameters for TermA.	CiscoRTPInputstartedEv CiscoRTPOutputStarted EvCallCtlTermConnTalkingEv	

Scenario Ten

Call from a terminal registered with registration type as CiscoBaseMediaTerminal .STATIC_MEDIA_REGISTRATION_FOR_GET_PORT_SUPPORT.

Action	Result	Call information
Application registers the terminal statically using the new CiscoBaseMediaTerminal .register() API and passes STATIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT for registration type and CiscoTerminal.IP _ADDRESSING_MODE_IPv4 for activeAddressingMode	CiscoTermInServiceEvterm ACiscoAddrInServiceEv A	
Application invokes connect() API to connect to another address B on terminal termB.	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TermA GC1 TermConnActiveEvTerm A GC1 CallCtlTermConnTalkingEv TermA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 CiscoMediaOpenIPPortEv TermA	getMediaIPAddressingMode() = IPv4 (((CiscoBaseMediaTerminal) (ev.getTerminal())). getRegistrationType = CiscoBaseMediaTerminal . passes STATIC_MEDIA_REGISTRATION_FORGET_PORT_SUPPORT
Application sets the RTP parameters.(IPv4 address and port). Application answers the call on B.	GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1 CallCtlTermConnRingingEvImpl B GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv CiscoRTPOutputStarted	

Scenario Eleven

Two Applications registering same CTIPort or RP with different values for registrationType (static).

Action	Result	Call information
Application1 registers the terminal TermA statically using the new register() API and STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	
Application2 registers the terminal TermA statically usig the new register() API and passes something other than STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	PlatformException	CTIERR_MEDIA_ALREADY_ TERMINATED_STATIC_ GETPORT_SUPPORT
Application3 registers the terminal TermA statically usig the new register() API and passes STATIC_MEDIA_REGISTRATION _FOR_GET_PORT_SUPPORT for registration type	CiscoTermInServiceEv termACiscoAddrInServiceEv A	

SRTP Key Material

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

Scenario One

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

Scenario Two

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

Scenario Three

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

Super Provider Message Flow

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

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

SuperProvider and Change Notification Enhancements Use Cases

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

Scenario One

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

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

Scenario Two

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

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

Scenario Three

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

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

Scenario Four

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

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

Scenario Five

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

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

Scenario Six

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

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

Scenario Seven

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

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

Support for Cisco Unified IP Phone 6901

Scenario 1

Phone A is a Cisco Unified IP Phone 6901 and phone B is a normal SCCP/SIP phone. Application is observing the devices A and B. Phone A is off-hook and application initiates a call through createCall() API from phone A to phone B.

Configuration:

- Phone A – Cisco Unified IP Phone 6901
- Phone B – SCCP/SIP Device

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	Application observes A and B.
A goes off-hook Application calls createCall() and call connect() API to Call B.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] GC1: TermConnDroppedEv [Term A] CallCtlTermConnDroppedEv [Term A] ConnDisconnectedEv A CallCtlConnDisconnectedEv A CallInvalidEv	

Action	Result	Call info
	GC2: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] CallCtlConnDialingEv A CallCtlConnEstablishedEv A ConnCreatedEv – B - ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv [Term B] TermConnRingingEv [Term B] CallCtlTermConnRingingEvImpl [Term B]	
B answers the call and A & B are connected.	GC2: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 2

Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application is observing both the devices A & B. User initiates a call from phone A to phone B. Phone B goes off-hook and answers the incoming call.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
B answers the call and A & B are connected.	GC2: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 3

Phone A is Cisco Unified IP Phone 6901 and phone B is a normal SCCP/SIP phone. Application is observing both the devices A and B. Phone A is on-hook and application initiates a call from phone A to phone B.

Configuration:

- Phone A – Cisco Unified IP Phone 6901
- Phone B – SCCP/SIP Device

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A is on-hook and application call createcall() and connect API to call B	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 4

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B is on-hook and application tries to answer the call on B.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A intitiates a call to B from application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B is on-hook and tries to answer the call from the application.	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 5

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Scenario 6

Call Park Reversion Timer is set to 30 seconds at service parameter page. Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call. B goes on-hook.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Action	Result	Call info
B goes on-hook. Call Park reversion Timer expires after 30 seconds and the call comes back to B. B tries to answer the call from the application.	GC1: ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv [Term B] TermConnRingingEv [Term B] CallCtlTermConnRingingEvImpl [Term B] Jtapi throwsEception: InvalidStateException.	Operation not available in current state.
B goes off-hook and answers the call.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 7

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application. C is off-hook and unparks the call from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL

Action	Result	Call info
C unparks the call from the application.	GC2: CallActiveEv ConnCreatedEv –C ConnConnectedEv – C CallCtlConnInitiatedEv - C - ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnOfferedEv ParkDN	
	GC1: ConnDisconnectedEv ParkDN CallCtlConnDisconnectedEv ParkDN	
	GC1: ConnCreatedEv –C ConnConnectedEv – C CallCtlConnEstablishedEv – C TermConnCreatedEv [Term C] TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C] ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnOfferedEv ParkDN	
	GC2: ConnDisconnectedEv ParkDN CallCtlConnDisconnectedEv ParkDN TermConnDroppedEv [Term C] CallCtlTermConnDroppedEv [Term C] ConnDisconnectedEv C GC2 CiscoCallChangedEv CallCtlConnDisconnectedEv C CallInvalidEv	

Scenario 8

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B parks the call from application. C is on-hook and unparks the call from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Action	Result	Call info
B parks the call from the application.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv ParkDN ConnInProgressEv ParkDN CallCtlConnQueuedEv ParkDN	currentCalling = A currentCalled = Park DN CAUSE = CAUSE_NORMAL
C unparks the call from the application.	Jtapi throws Exception: InvalidStateException	Operation not available in current state.

Scenario 9

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B transfers the call to C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B makes consult call to C	GC1: CallCtlTermConnHeldEv B GC2: CallActiveEv ConnCreatedEv –B ConnConnectedEv – B CallCtlConnInitiatedEv - BTermConnCreatedEv - [Term B] TermConnActiveEv –[Term B] CallCtlTermConnTalkingEv –[Term B] -	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL

Action	Result	Call info
C goes off-hook and answers the call. B and C are connected.	GC2: ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL
B completes transfer by invoking GC1.transfer(GC2)	GC1: CiscoTermConnSelectChangedEv [Term B] GC2: CiscoTermConnSelectChangedEv [Term B] GC1 CiscoTransferStartEv GC2 CiscoCallChangedEv	
	GC1: ConnCreatedEv - C ConnConnectedEv – C CallCtlConnEstablishedEv - C TermConnCreatedEv - [Term C] TermConnActiveEv –[Term C] CallCtlTermConnTalkingEv –[Term C]	
	GC2: TermConnDroppedEv [Term C] CallCtlTermConnDroppedEv [Term C] ConnDisconnectedEv C CallCtlConnDisconnectedEv C	

Action	Result	Call info
	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B GC2: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B CallInvalidEv CiscoTransferEndEv	

Scenario 10

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B does a conference with C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv – C	

Action	Result	Call info
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B makes consult call to C. C goes off-hook and answers the call.	GC1: CallCtlTermConnHeldEv B GC2: CallActiveEv ConnCreatedEv –B ConnConnectedEv – B - ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = B currentCalled = C CAUSE = CAUSE_NORMAL

Action	Result	Call info
B conferences two calls by invoking GC1.conference(GC2)	GC1: CiscoTermConnSelectChangedEv [Term B] GC2: CiscoTermConnSelectChangedEv [Term B] GC1 CiscoConferenceStartEv GC2 CiscoCallChangedEv	
	GC1: ConnCreatedEv - C ConnConnectedEv – C CallCtlConnEstablishedEv - C TermConnCreatedEv - [Term C] TermConnActiveEv –[Term C] CallCtlTermConnTalkingEv –[Term C] GC2: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B CallInvalidEv GC1 CiscoConferenceEndedEv	

Scenario 11

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901 models. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B redirects the call to C from the application.

Configuration:

- Phone A – SCCP/SIP Device
- Phones B and C – Aleta Device

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	

Action	Result	Call info
A intitiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B redirects the call to C from the application.	GC1: ConnCreatedEv C ConnInProgressEv C CallCtlConnOfferedEv C TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B	

Action	Result	Call info
C is off-hook and answers the call from application.	CallCtlConnDisconnectedEv B ConnAlertingEv C CallCtlConnAlertingEv C TermConnCreatedEv [Term C] TermConnRingingEv [Term C] CallCtlTermConnRingingEvImpl [Term C] ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = A currentCalled = C CAUSE = CAUSE_NORMAL

Scenario 12

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing the corresponding button from the phone. B resumes the call by pressing Line key from the phone.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Aleta Phone

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A initiates a call from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL

Action	Result	Call info
B answers the call, and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses Hold button from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses Line button from the phone and resumes the call.	GC1: CallCtlTermConnTalkingEv B	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 13

Application is observing both the devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing the button and goes on-hook. Now B tries to resume the call from application.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	

Action	Result	Call info
A initiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses hold hardkey from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	
B goes on-hook. B tries to resume the call from the application.	Exception: InvalidStateException.	Operation not available in current state.

Scenario 14

Application is observing devices A and B. Phone A is a normal SCCP/SIP phone and phone B is Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B puts the call on hold by pressing hard key and goes on-hook. Now B tries to resume the call by pressing the line hard key from phone.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv - B	
A intitiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A & B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B presses hold hardkey from the phone and puts the call on hold.	GC1: CallCtlTermConnHeldEv B	
B goes on-hook. B tries to resume the call by pressing the line key from phone.	GC1: CallCtlConnTalkingEv [Term B] TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B TermConnDroppedEv [Term A] CallCtlTermConnDroppedEv [Term A] ConnDisconnectedEv A CallCtlConnDisconnectedEv A CallInvalidEv	Operation not available in current state.

Scenario 15

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. CFA is set to C at phone B. Application initiates a call from phone A to phone B. Due to CFA set to C, call goes to C. C goes off-hook and answers the call.

Configuration:

- Phone A – SCCP/SIP Device
- Phone B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A intitiates a call to B from the application. B has CFA set to C. Call goes to C.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	CAUSE = CAUSE_NORMAL REASON = FORWARDALL
C goes off-hook and answers the call. A & C are connected.	GC1: ConnConnectedEv C CallCtlConnEstablishedEv C TermConnActiveEv [Term C] CallCtlTermConnTalkingEv [Term C]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL

Scenario 16

Application is observing the devices A and B. Phone C is not observed. Phone A is a normal SCCP/SIP phone, B and C are Cisco Unified IP Phone 6901. Application initiates a call from phone A to phone B. B goes off-hook and answers the call. B redirects the call to C. C goes off-hook and answers the call.

Configuration:

- Phone A – SCCP/SIP Device

- Phone B and C – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A and B. Phone C is not observed.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B	
A initiates a call to B from the application.	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	currentCalling = A currentCalled = null CAUSE = CAUSE_NORMAL
B answers the call and A and B are connected.	GC1: ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	currentCalling = A currentCalled = B CAUSE = CAUSE_NORMAL
B redirect the call to C. C goes off-hook and answers the call.	GC1: TermConnDroppedEv [Term B] CallCtlTermConnDroppedEv [Term B] ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv C ConnAlertingEv C CallCtlConnAlertingEv C ConnConnectedEv C CallCtlConnEstablishedEv C	

Scenario 17

Application is observing the devices A, B and C. Phone A is a normal SCCP/SIP phone, B is Cisco Unified IP Phone 6901 and phone C is a normal SCCP/SIP phone. B and C have a shared line. Application initiates a call from phone A to shared line on B and C. B goes off-hook and answers the call.

Configuration:

- Phone A and C – SCCP/SIP Device
- Phone B – Cisco Unified IP Phone 6901

Action	Result	Call info
Application observes A, B and C.	CiscoAddrInServiceEv – A CiscoAddrInServiceEv – B CiscoAddrInServiceEv - C	
A initiates a call to the shared line on B and C	GC1: CallActiveEv ConnCreatedEv –A ConnConnectedEv – A CallCtlConnInitiatedEv - ATermConnCreatedEv - [Term A] TermConnActiveEv –[Term A] CallCtlTermConnTalkingEv –[Term A] -	
B goes off-hook and answers the call.	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnCreatedEv [Term C] TermConnPassiveEv [Term C] CallCtlTermConnInUseEv [Term C] TermConnActiveEv [Term B] CallCtlTermConnTalkingEv [Term B]	

SHA Support for Digital Signatures

The following tables display the CallInfo messages for the following three use cases:

- SHA-1 is the configured encryption algorithm

- SHA-512 is the configured encryption algorithm (Cisco JTAPI is version 11.5)
- SHA-512 is the configured encryption algorithm (Cisco JTAPI is a pre-11.5 version)

Table 89: SHA-1 is Configured

Action	CallInfo
TFTP File Signature Algorithm enterprise parameteris set to **SHA1** (the default value). JTAPIProperties.setSecurityPropertyForInstance() is invoked with CAPF login and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=true

Table 90: SHA-512 is Configured (Cisco JTAPI is version 11.5)

Action	CallInfo
TFTP File Signature Algorithm enterprise parameter is set to **SHA 512**. JTAPIProperties.setSecurityPropertyForInstance() is invoked with username and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=true

Table 91: SHA-512 is Configured (Cisco JTAPI is pre-11.5 version)

Action	CallInfo
TFTP File Signature Algorithm enterprise parameter is set to **SHA 512**. JTAPIProperties.setSecurityPropertyForInstance() is invoked with username and instance ID.	JTAPIProperties.getSecurityPropertyForInstance(). certificateStatus=false

TLS Security

Message flow for updating certificate and establishing TLS certificate is illustrated in the following two figures.

Figure 3: CTI/API Security Approach

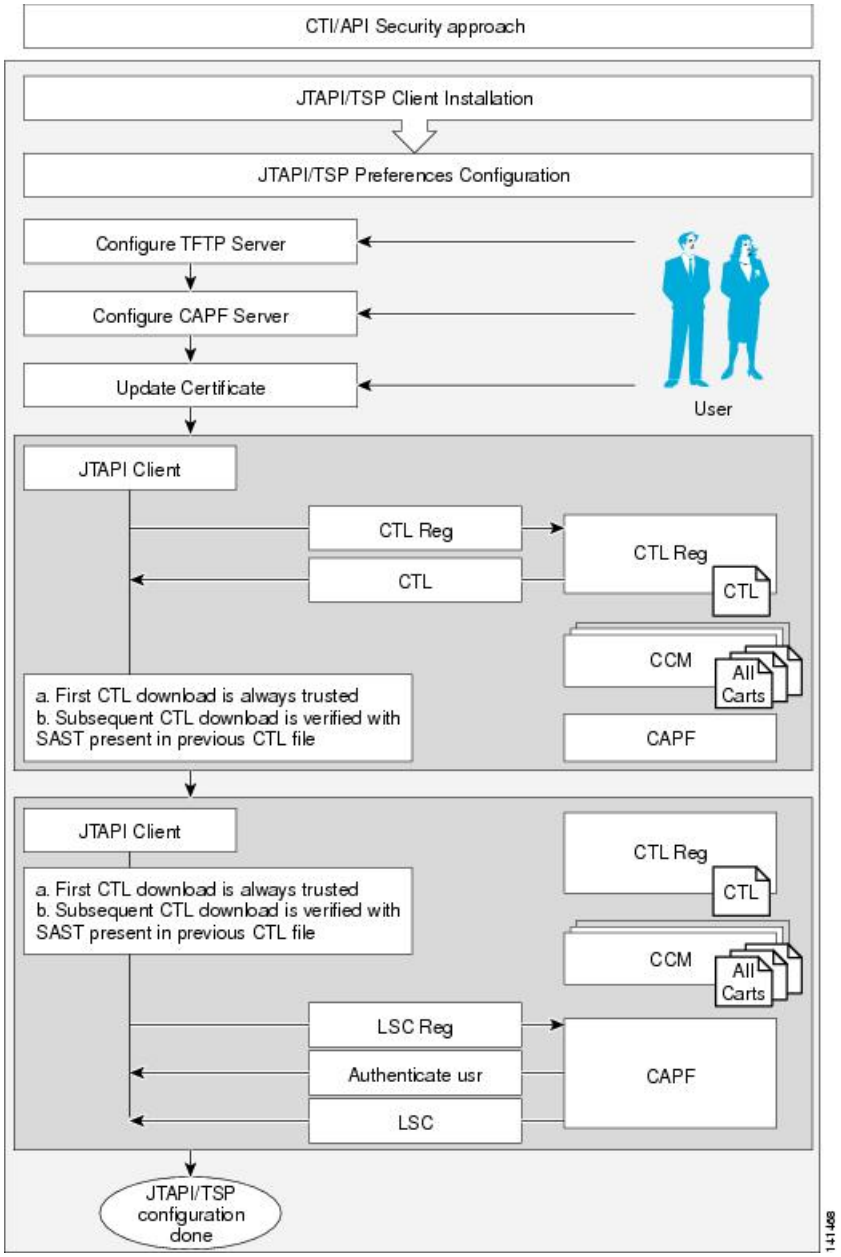
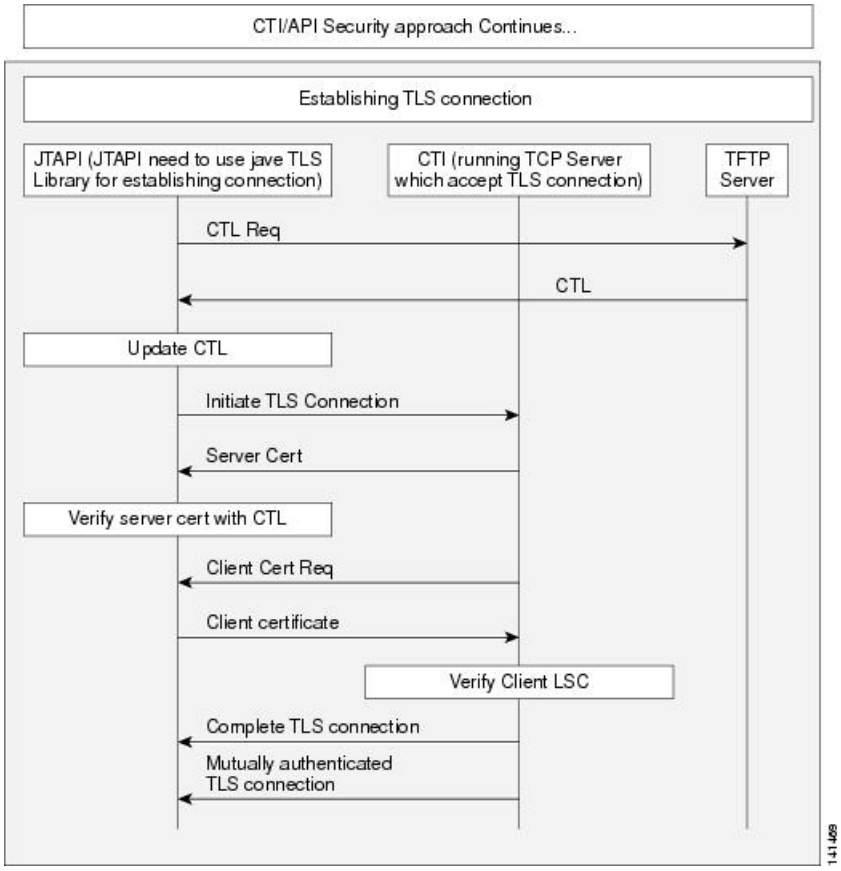


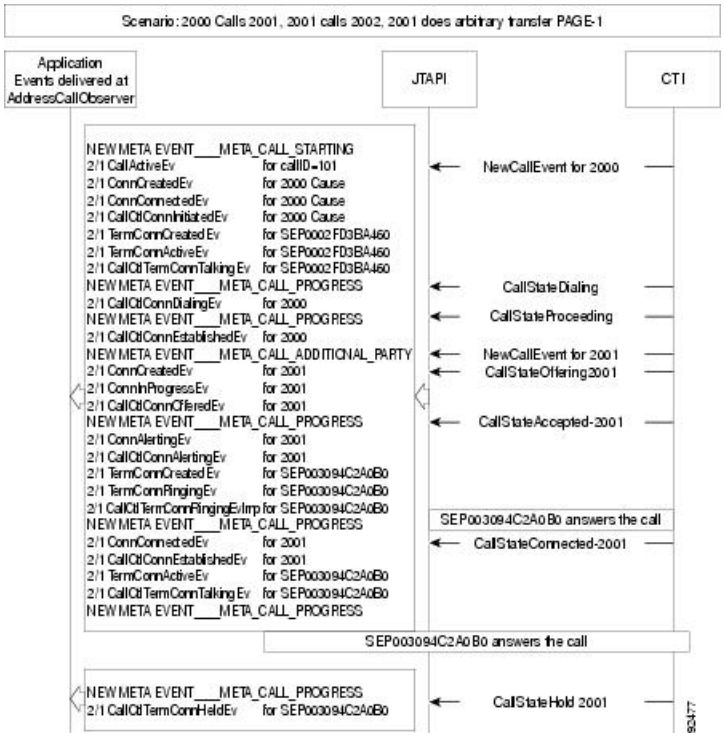
Figure 4: CTI/API Security Approach (Continued)



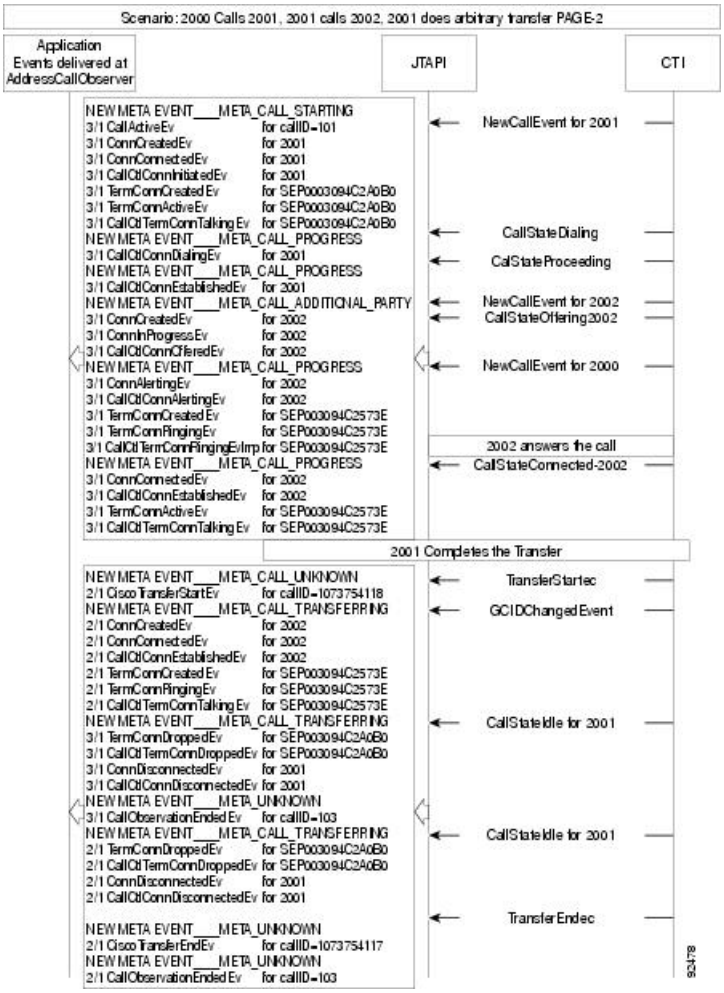
Transfer and Direct Transfer

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

DirectTransfer/Arbitrary Transfer Scenario



Direct Transfer/Arbitrary Transfer-Page 2



Consult Transfer

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

Unicode Support

Unicode Display Name Scenario

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

GetLocale and UniCodeCapabilities of Terminal

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

Unrestricted Unified CM

Use Case One

The application tries to register with an insecure CTI Port to the unrestricted Cisco Unified Communications Manager.

Action	Result	Call information
Application opens a provider with the unrestricted Cisco Unified Communications Manager and tries to register with an insecure phone CTI Port 'A'.	[Term A] CiscoTermOutOfService[A] CiscoAddrOutOfServiceEv[Term A] CiscoTermInServiceEv[A] CiscoAddrInServiceEv	

Variance

Application tries to register with an insecure route point to the unrestricted Cisco Unified Communications Manager.

Use Case Two

Restricted Cisco Unified Communications Manager is upgraded to unrestricted Cisco Unified Communications Manager. The application tries to register with a secure phone after the upgrade.

In some of the scenarios, where the application registers a device in a secure mode, the registeration is successful but eventually can be rejected with a new error code - CiscoTermRegisterationFailedEv.

Variance

Application tries to register a secure Route Point after an upgrade from a restricted Cisco Unified Communications Manager to unrestricted Cisco Unified Communications Manager.

Video Capabilities and Multi-Media Information

The following sections describe use cases that are related to video capabilities and multi-media information feature.

Scenario One

Phone A is video capable, telepresence capable, with 1 screen and a camera, and in registered state. User1 has phone A in the control list. User queries for multimedia capabilities before adding a terminal observer.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getVid eoCapability() on termA		termA. getCiscoMulti MediaCapabilityInfo() .getVideoCa pability() = CiscoMultiMedia CapabilityInfo.ENABLED
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getTel epresenceInfo () on termA		termA. getCiscoMulti MediaCapabilityInfo() .getTelepres enceInfo () = CiscoMultiMedia CapabilityInfo.TELEPRESENC EINTEROP_ENABLED
User1 invokes CiscoTerminal.i getCiscoMulti MediaCapabilityInfo() .getScr eenCount () on termA		termA. getCiscoMulti MediaCapabilityInfo() .getScreenC ount () = 1

Scenario Two

Phone A is a video disabled SIP Phone (In Cisco Unified CM Administration Phone page, Video Capabilities field is “Disabled”). Phone A is in a registered state.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA.getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Enabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. ENABLED
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. ENABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Disabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCa pability() = CiscoMultiMediaCapabilityInfo. DISABLED

Scenario Three

Phone A is a video disabled SCCP Phone (In Cisco Unified CM Administration Phone page, Video Capabilities field is “Disabled”). Phone A is in a registered state.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED

Action	Events	Call Info
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Enabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. ENABLED
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. ENABLED
In Device Configuration Cisco Unified CM Administration pages- Video Capabilities field is changed to “Disabled”	CiscoProvTerminalMulti MediaCapabilityChangedEv will not be delivered to applications, as the device will unregister and register back. In this case applications can query video capability after the device is registered.	termA. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo. DISABLED

Scenario Four

Phone A is video capable, telepresence capable, has 1 screen, has a camera, and is in an unregistered state. User1 has phone A in the control list. User queries for multimedia capabilities before adding a terminal observer.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getVideoCapability() on termA		InvalidStateException: Terminal is not in registered state.
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getTelepresenceInfo() on termA		InvalidStateException: Terminal is not in registered state.
User1 invokes CiscoTerminal. getCiscoMultiMedia CapabilityInfo().getScreenCount() on termA		InvalidStateException: Terminal is not in registered state.

Scenario Five

Phone A is video capable, telepresence capable, with 1 screen and a camera. User1 has phone A in the control list. Application queries for mutlimedia capabilities on CiscoTerminal.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA	CiscoTermOutOfServiceEv CiscoTermInServiceEv	termA. getCiscoMulti MediaCapabilityInfo().getVideoCapability() = NONE
User1 invokes CiscoTerminal. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getVideoCapability() = CiscoMultiMediaCapabilityInfo.ENABLED
User1 invokes CiscoTerminal. i getCiscoMulti MediaCapabilityInfo(). getTelepresenceInfo() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getTelepresenceInfo() = CiscoMultiMediaCapabilityInfo.TELEPRESENCEINTEROP_ENABLED
User1 invokes CiscoTerminal. i getCiscoMulti MediaCapabilityInfo(). getScreenCount() on termA		termA. getCiscoMulti MediaCapabilityInfo(). getScreenCount () = 1

Scenario Six

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registers it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Action	Events	Call Info
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario Seven

Phone A is a CTI Port or RoutePoint. User1 has phone A in the control list. The user invokes CiscoTerminal.getCiscoMultiMediaCapabilityInfo().getVideoCapability().

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 opens termA and registers it	CiscoTermOutOfServiceEv CiscoTermInServiceEv	
User1 invokes CiscoTerminal. getCiscoMultiMediaCapabilityInfo(). getVideoCapability() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getTelepresenceInfo() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals
User1 invokes CiscoTerminal. i getCiscoMultiMediaCapabilityInfo(). getScreenCount() on termA		The API returns MethodNotSupportedException - Not supported on Media Terminals and RPs and Remote Terminals

Scenario Eight

Basic Video call: Phone A is video enabled, Telepresence Enabled and has 1 screen. Phone B has video disabled, Telepresence Disabled and has 0 screens. Both phones are in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscoTermInServiceEv TermA CiscoTermInServiceEv TermB	

Action	Events	Call Info
User1 adds call observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEvImpl B	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresence capable device(CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getCallingTerminal VideoCapability() on GC1.		The API returns 0, indicating video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getTelepresenceInfo() on GC1.		The API returns 0, indicating telepresence capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 01 screen, for TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo() .getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresence capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ENABLED) for TermA
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getCallingTerminal VideoCapability() on GC1.		The API returns 0, indicating not video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo().getTelepresenceInfo() on GC1.		The API returns 0, indicating device is not telepresence capable (TELEPRESENCEINTEROP_DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 0 screens, for TermB

Scenario Nine

Shared Line: Phone A has video enabled, Phone B has video disabled and Phone B' has video enabled. B' is in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone B'	CiscotermInServiceEv TermB'	
User1 adds call observes on the address B'	CiscoAddrInServiceEv B'	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo().getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. DISABLED) for TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo().getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		Application will receive the following incorrect data: The API returns 1, indicating not video capable device(CiscoMultiMediaCapabilityInfo. ENABLED) for Term B.

Scenario Ten

Shared Line: Phone A has video enabled, Phone B has video disabled and Phone B' has video enabled. B and B' is in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone B and Phone B'	CiscotermInServiceEv TermB CiscotermInServiceEv TermB'	
User1 adds call observes on the address B and B'	CiscoAddrInServiceEv B CiscoAddrInServiceEv B'	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.).getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo. .getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. DISABLED) for TermB'
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.).getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.).getCallingTerminalVideoCapability() on GC1.		Application will receive the following incorrect data: The API returns 1, indicating not video capable device(CiscoMultiMediaCapabilityInfo. ENABLED) for Term B.

Scenario Eleven

Basic Video call: Phone A is video enabled, Telepresence enabled and has 1 screen. Phone B has video disabled, Telepresence disabled and has 0 screens. Phone A is in User1's control list, Phone A is observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns -1, indicating video capability is not known (UKNOWN) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns -1, indicating telepresence capability is not known (UKNOWN) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns -1, indicating screen count capability is not known TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns -1, indicating video capability is not known (UKNOWN) for TermB

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns -1, indicating telepresence capability is not known (UKNOWN) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns -1, indicating screen count capability is not known TermB
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMediaCapabilityInfo. ENABLED) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns 1, indicating telepresenc capable device (CiscoMultiMedia CapabilityInfo. TELEPRESENCEINTEROP_ ENABLED) for TermA
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating device has 1 screen, for TermA
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for Term B.

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1.		The API returns 0, indicating device is not telepresence capable (TELEPRESENCEINTEROP_ DISABLED) for TermB
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 0, indicating device has 0 screens, for TermB

Scenario Twelve

MultiMedia Streams: Phone A and B have video enabled, and both phones are in User1's control list.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionAddress() on Terminal A		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionPort() on Terminal A		The API returns port number from which media will be directed.

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionAddress() on Terminal A		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionPort() on Terminal A		The API returns the payload format.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getPayloadType() on Terminal A		The API returns the maximum bit rate.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getMaxBitRate() on Terminal A		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal A		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaType() on Terminal A		
CiscoMultiMediaStreamsInfoEv. getProperties(). isKeyInfoPresent() on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal A		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal A		The API returns 3(MEDIA_ENCRYPTED_KEYS _UNAVAILABLE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionAddress() on Terminal B		The API returns IP address to which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getReceptionPort() on Terminal B		The API returns port number to which media will be directed.

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionAddress() on Terminal B		The API returns IP address from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getTransmissionPort() on Terminal B		The API returns port number from which media will be directed.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getPayloadType() on Terminal B		The API returns the payload format.
CiscoMultiMediaStreamsInfoEv. getProperties(). getRTPProperties(). getMaxBitRate() on Terminal B		The API returns the maximum bit rate.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaType() on Terminal B		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties(). isKeyInfoPresent() on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal B		The API returns 3(MEDIA_ENCRYPTED_KEYS _UNAVAILABLE)
B holds the call.		
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal A		The API returns 3(INACTIVE)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 3(INACTIVE)
B unholds the call.		
CiscoMultiMediaStreamsInfoEv. getCallingTerminalVideoCapability() on GC1.		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)

Scenario Thirteen

Redirect: Phone A, B, and C have video enabled, and A, B phones are in User1's control list, and Phone C is in User2's control list.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B User2 adds terminal observers on Phone C.	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC	
User1 adds callObserves on the address A and B User2 adds callObserves on the address C	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal A		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties().getMultiMediaType() on Terminal A		The API returns 2(MAIN_VIDEO)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties().isKeyInfoPresent() on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal B		The API returns 0(TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties().getMultiMediaType() on Terminal B		The API returns 2(MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties().isKeyInfoPresent() on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaEncryptionKeyInfo() on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties(). getMultiMediaSecurityIndicator() on Terminal B		The API returns 2(MAIN_VIDEO) 3(MEDIA_ENCRYPTED _KEYS_UNAVAILABLE)
B redirects the call to C. C answers		
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal B		The API returns 3(INACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal C		The API returns 0(ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties() .getMultiMediaConnectionMode() on Terminal A		The API returns 0(ACTIVE)

Action	Events	Call Info
App does CiscoCall. getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capability is enabled (CiscoMultiMediaCapabilityInfo.ENABLED) for TermA (far-end party).
App does CiscoCall. getCalledTerminalMult iMediaCapabilityInfo() on GC2.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).
CiscoMultiMediaStreamsInfoEv. getCalledTerminalMult iMediaCapabilityInfo on GC2.		The API returns 1, indicating video capability is enabled (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).

Scenario Fourteen

Transfer: Phone A, B and C have video enabled, and A, B phones are in User1's control list and C is in User2's control list, A, B and C are in cluster1.

Action	Events	Call Info
User1 Opens Provider and adds observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B User2 adds terminal observers on Phone C.	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC	
User1 adds callObserves on the address A and B User2 adds callObserves on the address C	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal A		The API returns 0 (TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal A		The API returns 2 (MAIN_VIDEO)

Action	Events	Call Info
CiscoMultiMediaStreamsInfoEv. getProperties ().isKeyInfoPresent () on Terminal A		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal B		The API returns 0 (TRANSMIT_AND_RECEIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal B		The API returns 2 (MAIN_VIDEO)
CiscoMultiMediaStreamsInfoEv. getProperties ().isKeyInfoPresent () on Terminal B		The API returns False
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaEncryptionKeyInfo () on Terminal B		The API returns NULL.
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaSecurityIndicator () on Terminal B		The API returns 3 (MEDIA_ENCRYPTED_ KEYS_UNAVAILABLE)
B does a consult call to C. C answers		
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaType () on Terminal B		The API returns 0 (ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal C		The API returns 0 (ACTIVE)
App does CiscoCall. getCalledTerminalMultiMediaCapabilityInfo () on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB (far-end party).

Action	Events	Call Info
B does GC1.transfer (GC2). C answers		
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal B		The API returns 3 (INACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal C		The API returns 0 (ACTIVE)
CiscoMultiMediaStreamsInfoEv. getProperties ().getMultiMediaConnectionMode () on Terminal A		The API returns 0 (ACTIVE)
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfoon GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo () on GC2.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).
App does CiscoCall. getCalledTerminalMultiMediaCapabilityInfo ().getCallingTerminalVideoCapability () onGC1.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC (far-end party).
App does CiscoCall. getCallingTerminalMultiMediaCapabilityInfo () on GC1.		The API returns 1, indicating video capability is known (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA (far-end party).

Scenario Fifteen

Negotiated video capability will be reported to the called party across a cluster call (over SIP – ICT trunk) using early offer (Phone A is a video disabled SIP Phone in cluster 1 and Phone B is a video enabled SIP Phone in cluster 2). User1 has Phone A in the control list, Phone A is observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	

Action	Events	Call Info
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds callObserves on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermA.

Action	Events	Call Info
App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermB.
Variation 1: A and B are SIP Phone and have video enabled. User1 makes a call from A to B. B answers the call. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device(CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.

Scenario Sixteen

Multiple redirects over SIP trunk (A, B, C and D are video enabled SIP Phones, A is in cluster 1 and B, C, D are in cluster 2). Phone A, B, C and D are in User1's control list, Phone A, B, C and D are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A B, C, and D	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC CiscotermInServiceEv TermD	
User1 adds callObserves on the address A, B, C, and D	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C CiscoAddrInServiceEv D	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.
App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB.

Action	Events	Call Info
B redirects the call to C.		
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
C redirects the call to D.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTermina lMultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device(CiscoMultiMedia CapabilityInfo.ENABLED) for TermD.

Scenario Seventeen

Redirect over SIP trunk (Phone A is video enabled SIP Phone, and Phone B and C have video disabled. Phone A is in cluster 1 and Phone B and C are in cluster 2). Phone A and B are in User1's control list and Phone C is in User2's control list, Phone A, B and C are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermA.
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo. DISABLED) for TermC.

Action	Events	Call Info
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo. ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo. ENABLED) for TermC.

Scenario Eighteen

Shared Line – Hold and Resume scenario over SIP trunk (Phone A and C are video enabled SIP Phones and Phone B is video disabled, Phone A is in cluster 1 and Phone B and C are in cluster 2. Phone B and C are shared lines). Phone A and B are in User1's control list, Phone A and B are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
B redirects the call to C. C answers		
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo() .getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo() .getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.

Action	Events	Call Info
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo() .getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo() .getCallingTerminal VideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermC.

Scenario Nineteen

Multiple redirects over H323 ICT trunk (A, B, C and D are video enabled SIP Phones, A is in cluster 1 and B, C, D are in cluster 2). Phone A, B, C and D are in User1's control list, Phone A, B, C and D are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A B, C, and D	CiscotermInServiceEv TermA CiscotermInServiceEv TermB CiscotermInServiceEv TermC CiscotermInServiceEv TermD	
User1 adds call Observes on the address A B, C, and D	CiscoAddrInServiceEv A CiscoAddrInServiceEv B CiscoAddrInServiceEv C CiscoAddrInServiceEv D	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermB. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermB. The API returns -1, indicating screen count isUNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermB.
B redirects the call to C. C answers		

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.
C redirects the call to D.		

Action	Events	Call Info
App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo.getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermD. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.

Scenario Twenty

Redirect over H323 trunk (Phone A is video enabled SIP Phone and Phone B and C have video disabled, Phone A is in cluster 1 and Phone B and C are in cluster 2). Phone A and B are in User1's control list and Phone C is in user2's control list, Phone A, B and C are observed.

Action	Events	Call Info
User1 Opens Provider and adds a provider observer	ProvInServiceEv	
User1 adds terminal observers on Phone A and Phone B	CiscotermInServiceEv TermA CiscotermInServiceEv TermB	
User1 adds call Observes on the address A and B	CiscoAddrInServiceEv A CiscoAddrInServiceEv B	

Action	Events	Call Info
User1 makes a call from A to B	GC1: CallActiveEv GC1: ConnCreatedEv A GC1:ConnConnectedEv A GC1:CallCtlConnInitiatedEv A GC1:TermConnCreatedEv TermA GC1:TermConnActiveEv TermA GC1:CallCtlTermConnTalkingEv TermA GC1:CallCtlConnDialingEv A GC1CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 ConnInProgressEv B GC1 CallCtlConnOfferedEv B GC1 ConnAletingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv B GC1 TermConnRingingEv B GC1CallCtlTermConnRingingEv Impl B	
B answers the call	GC1 ConnConnectedEv B GC1 CallCtlConnEstablishedEv B GC1 TermConnActiveEv B GC1 CallCtlTermConnTalkingEv B CiscoRTPInputStartedEv TermA CiscoRTPInputStartedEv TermB CiscoRTPOutputStartedEv TermA CiscoRTPOutputStartedEv TermB	
B redirects the call to C. C answers		
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1.		The API returns 1, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA.

Action	Events	Call Info
App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 0, indicating not a video capable device (CiscoMultiMedia CapabilityInfo.DISABLED) for TermC.
Variation 1: A and B have video enabled, C has video disabled App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermC.
C redirects the call to D.		

Action	Events	Call Info
App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getVideoCapability() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getCallingTerminalVideoCapability() on GC1. App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCalledTerminal MultiMediaCapabilityInfo(). getTelepresenceInfo() on GC1. App does CiscoCall. getCallingTerminal MultiMediaCapabilityInfo.getScreenCount() on GC1. App does CiscoCall. getCalledTerminalMulti MediaCapabilityInfo. getScreenCount() on GC1.		The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermA. The API returns 1, indicating video capable device (CiscoMultiMedia CapabilityInfo.ENABLED) for TermD. The API returns -1, indicating telepresence is UNKNOWN for TermA. The API returns -1, indicating telepresence is UNKNOWN for TermC. The API returns -1, indicating screen count is UNKNOWN for TermA. The API returns -1, indicating screen count is UNKNOWN for TermC.

Video On Hold

Pre-conditions to all video on hold use cases, unless specified otherwise:

- Provider is in IN_SERVICE state
- All addresses and terminals are already in service.
- Device A (IP Phone – Name: “SEP2401C7824EA3”, Line A1 (dn: 9000))
- Device B (IP Phone – Name: “SEP2401C7824EAE”, Line B1 (dn: 9001))
- The content id corresponding to VoH stream is contentID1.
- User1 has in its control list: Devices A and B. All devices and lines are observed.

Scenario One

Invoke hold() on basic call between two ip phones to stream video to the held party

Action	Events	Call information / Notes
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes Callconnect("SEP2401C7824EA3", "9000", "9001").	GC1: CallActiveEv GC1: ConnCreatedEv 9000 GC1: ConnConnectedEv 9000 GC1: CallCtlConnInitiatedEv 9000 GC1: TermConnCreatedEv SEP2401C7824EA3 GC1: TermConnActiveEv SEP2401C7824EA3 GC1: CallCtlTermConnTalkingEv SEP2401C7824EA3 GC1: CallCtlConnDialingEv 9000 GC1: CallCtlConnEstablishedEv 9000 GC1: ConnCreatedEv 9001 GC1: ConnInProgressEv 9001 GC1: CallCtlConnOfferedEv 9001 GC1: ConnAlertingEv 9001 GC1: CallCtlConnAlertingEv 9001 GC1: TermConnCreatedEv SEP2401C7824EAE GC1: TermConnRingingEv SEP2401C7824EAE GC1: CallCtlTermConnRingingEv SEP2401C7824EAE	
Device B answers the call.	GC1: ConnConnectedEv 9001 GC1: CallCtlConnEstablishedEv 9001 GC1: TermConnActiveEv SEP2401C7824EAE GC1: CallCtlTermConnTalkingEv SEP2401C7824EAE	
User1 invokes hold("contentID1") on terminal connection of Device A.	GC1: CallCtlTermConnHeldEv SEP2401C7824EA3 **Note** You are able to see video streamed to the device that is placed on hold.	

Action	Events	Call information / Notes
A disconnects the call.	GC1: TermConnDroppedEv SEP2401C7824EA3 GC1: CallCtlTermConnDroppedEv SEP2401C7824EA3 GC1: ConnDisconnectedEv 9000 GC1: CallCtlConnDisconnectedEv 9000 GC1: TermConnDroppedEv SEP2401C7824EAE GC1: CallCtlTermConnDroppedEv SEP2401C7824EAE GC1: ConnDisconnectedEv 9001 GC1: CallCtlConnDisconnectedEv 9001 GC1: CallInvalidEv GC1: CallObservationEndedEv	

Verification Involving PSTN Reachability

Scenario 1

A calls B in other cluster (Normal Call); application is observing A

Action	Result	Call info
A dials B, B rings.	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	

Scenario 2

A calls B in other cluster (VIPR Call - Call gets routed through IP trunk); application is observing A

Action	Result	Call info
A dials B, B rings	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	

Scenario 3

A calls B, within same cluster. B redirects the call to external party C, the redirected call goes through IP Trunk due to VIPR feature; application is observing both A and B.

Action	Result	Call info
A calls B withis same cluster	GC1: CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A ConnCreatedEv B CallCtlConnOfferedEv B ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv TB TermConnRingingEv TB CallCtlTermConnRingingEv TB	

Action	Result	Call info
B redirects the call to external party C	GC1: TermConnDroppedEv TB CallCtlTermConnDroppedEv TB ConnDisconnectedEv B CallCtlConnDisconnectedEv B ConnCreatedEv C ConnConnectedEv C CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	CiscoFeatureReason = CiscoFeatureReason.REASON_REDIRCT CallCtlCause = CallCtlEv.CAUSE_REDIRECTED CiscoFeatureReason.REASON_REDIRECT

Scenario 4

A calls external party B; call goes through IP trunk but later call quality degrades and VIPR PSTN Fallback happens; application is observing A.

Action	Result	Call info
A dials B(VIPR Call), B rings	GC1: CallActiveEv ConnCreatedEv -A ConnConnectedEv - A CallCtlConnDialingEv - A TermConnCreatedEv - TA TermConnActiveEv -TA CallCtlTermConnTalkingEv - TA ConnCreatedEv B ConnConnectedEv B CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	
B answers	TA: CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	

Action	Result	Call info
Call Quality Degrades and call falls back to PSTN network	TA: CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	

Scenario 5

A calls B, B transfers the call to external Party C; Transferred call goes through IP trunk due to VIPR feature; application is observing both A and B.

Action	Result	Call info
A calls B; B answers	GC1 CallActiveEv GC1 ConnCreatedEv A GC1 ConnConnectedEv A GC1 CallCtlConnInitiatedEv A GC1 TermConnCreatedEv TA GC1 TermConnActiveEv TA GC1 CallCtlTermConnTalkingEv TA GC1 CallCtlConnDialingEv A GC1 CallCtlConnEstablishedEv A GC1 ConnCreatedEv B GC1 CallCtlConnOfferedEv B GC1 ConnAlertingEv B GC1 CallCtlConnAlertingEv B GC1 TermConnCreatedEv TB GC1 TermConnRingingEv TB GC1 CallCtlTermConnRingingEvImpl TB GC1: CallCtlConnEstablishedEv for A GC1: ConnConnectedEv for B GC1: CallCtlConnEstablishedEv for B	

Action	Result	Call info
B makes a consult call to external party C.	GC2: ConsultCallActiveEv GC2: ConnCreatedEv B GC2: ConnConnectedEv B GC2: CallCtlConnInitiatedEv B GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB GC2: CallCtlConnEstablishedEv B GC2: ConnCreatedEv C GC2: ConnConnectedEv for C GC2: CallCtlConnNetworkReachedEv GC2: CallCtlConnNetworkAlertingEv	
B completes the transfer; Two calls are transferred; A and C get connected over IP trunk due to VIPR feature	GC1 CiscoTermConnSelectChangedEv B GC2 CiscoTermConnSelectChangedEv B GC1 CiscoTransferStartedEv GC2 CiscoCallChangedEv GC1 ConnCreatedEv C GC1: ConnConnectedEv for C GC2 ConnDisconnectedEv for C GC2 CallCtlConnDisconnectedEv C GC1 TermConnDroppedEv B GC1 CallCtlTermConnDroppedEv B GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectecEv B GC2 TermConnDroppedEv B GC2 CallCtlTermConnDroppedEv B GC2 ConnDisconnectedEv B GC2 CallCtlConnDisconnectecEv B GC2 CallInvalidEv GC1 CiscoTransferEndEv	CiscoFeatureReason = CiscoFeatureReason. REASON_TRANSFEREDCALL

Scenario 6

A calls B, within the same cluster. B has CFA set to Forward all the calls to external party C, the forwarded call goes through IP Trunk due to VIPR feature. Application is observing A.

Action	Result	Call info
A calls B within the same cluster	GC1: CallActiveEv ConnCreatedEv A ConnConnectedEv A CallCtlConnInitiatedEv A TermConnCreatedEv TA TermConnActiveEv TA CallCtlTermConnTalkingEv TA CallCtlConnEstablishedEv A	
Call at B gets forwarded to external party C	GC1: CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv ConnCreatedEv C ConnConnectedEv C CallCtlConnNetworkReachedEv CallCtlConnNetworkAlertingEv	CiscoFeatureReason = CiscoFeatureReason.REASON_FORWARDALL CallCtlCause = CallCtlEv.CAUSE_REDIRECTED

Whisper Coaching

Use Case One - Monitoring with Mode as WHISPER

Start and Stop Whisper monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (CI1). B calls start monitor using GC2. The application has a call observer on both A and B. The monitoring capability enabled.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X… GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA in GC1 and mode as Whisper	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.WHISPER_MONITORGC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TBCiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A
A puts the call with X on hold	GC1: CallCtlTermConnHeldEv TA GC2:CiscoRTPInputStoppedEv TB GC2: CiscoRTPOutputStoppedEv TB GC1: CiscoRTPOutputStoppedEv TA GC1: CiscoRTPInputStoppedEv TA	

Action	Events	Call information
A resumes the call	GC1: CallCtlTermConnTalking TA GC1: CiscoRTPOutputStartedEv TA GC1: CiscoRTPInputStartedEv TA GC2: CiscoRTPOutputStartedEv TB GC2:CiscoRTPInputStartedEv TB	
B calls drop() on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv TB GC2: TermConnDroppedEv TB GC2: ConnDisconnectedEvB GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Two - Snapshot Use Case for Whisper Monitoring

- A is monitor target. B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Another application adds call observer on A after monitoring session is established.

Action	Events	Call information
	CallActiveEv for callID = GC1 Cause: CAUSE_SNAPSHOT GC1:ConnCreatedEv for A Cause: CAUSE_SNAPSHOT GC1:ConnConnectedEv for A Cause: CAUSE_SNAPSHOT GC1: ConnConnectedEv X GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_SNAPSHOT GC1: CiscoTermConnMonitoringStartEv TA Cause: CAUSE_SNAPSHOT getMonitorType() = CiscoCall.WHISPER_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_SNAPSHOT address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

- A is monitor target and B is monitor initiator. Caller X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Another application adds call observer on B after monitoring sessions are established.

Action	Events	Call information
	GC2:CallActive Cause: CAUSE_SNAPSHOT	GC1:
	GC2: ConnCreatedEv for B Cause: : CAUSE_SNAPSHOT	Calling: B
	GC2: ConnConnectedEv B	Called: A
	GC2: CallCtlConnEstablishedEv for B Cause: : CAUSE_SNAPSHOT	LRP: null
	GC2: TermConnCreatedEv TB	Current calling: B
	GC2: TermConnActiveEv TB	Current called:A
	GC2: CallCtlTermConnTalkingEv TB B Cause: : CAUSE_SNAPSHOT	
	GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET	
	GC2: ConnConnectedEv A	
	GC2: CallCtlConnEstablishedEv A	
	GC2:CiscoTermConnMonitorTargetInfoEv TB	
	Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1	
	CiscoMonitorInitatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
	[Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.]	

Use Case Three

Start Silent Monitoring then update the monitorType to Whisper mode followed by redirect and the updateMonitorType back to Silent monitor.

Whisper monitor: A is monitor target, B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2(mode = silent). Application has call observer on both A and B. Application has monitoring capability enabled. App updates the monitor mode to Whisper. B redirects the monitoring call to observed party C. App updates the monitor mode to silent and later drops monitoring call to stop monitoring.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL	GC1:
	GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL	Calling: X
	GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL	Called: A
	GC1: ConnConnectedEv X	LRP: null
	GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL	Current calling: X
	GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	Current called:A
	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as Silent	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.SILENT_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.SILENT_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.SILENT_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A
The application calls updateMonitorType() API on TermConn of B with mode as Whisper.	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv TB getMonitorType() = CiscoCall.WHISPER_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStartedEv TB	

Action	Events	Call information
B redirects the call to C and C answers.	GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnOfferedEv C GC2: TermConnCreatedEv TC GC2: TermConnActiveEv TC GC2: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv TB GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnectedEv TB GC2: CallCtlConnDisconnectedEv TB GC2: CiscoTermConnMonitorTargetInfoEv TC CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR GC2; CiscoTermConnMonitorInitatorInfoEv TA Cause: CAUSE_NORMAL address:C, device name: TC CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
The application calls updateMonitorType() API on TermConn of C with mode as Silent	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv TC getMonitorType() = CiscoCall.SILENT_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStoppedEv TC	
C calls drop on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv TC GC2: TermConnDroppedEv TC GC2: ConnDisconnectedEv C GC2: CallCtlConnDisconnectedEv C GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Four

Secured Whisper Monitoring followed by redirect to non-secured supervisor

Whisper monitor: A is monitor target, B is monitor initiator. Both A and B are Encrypted devices. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2(mode = whisper). Application has call observer on both A and B. Application has monitoring capability enabled. B redirects to observed party C which is non-secured.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as whisper	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TA getMonitorType() = CiscoCall.WHISPER_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
B redirects the call to C and C answers	GC2: ConnCreatedEv C GC2: ConnConnectedEv C GC2: CallCtlConnOfferedEv C GC2: TermConnCreatedEv TC GC2: TermConnActiveEv TC GC2: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv TB GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnectedEv TB GC2: CallCtlConnDisconnectedEv TB GC2: CiscoTermConnMonitorTargetInfoEv TC Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC2: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:C, device name: TC CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR GC2: ConnFailedEv C GC2: CallCtlConnFailedEv C cause : CAUSE_BCNAUTHORISED GC2: CallInvalidEv GC2: CiscoAddrMonitorTerminatedEv B cause : CAUSE_BCNAUTHORISED	

Use Case Five

Agent greeting is played and the application initiates the monitoring request (Silent/Whisper).

JTAPI throws InvalidStateException with error "CTIERR_RESOURCE_NOT_AVAILABLE" when the application sends a monitor request and when an agent greeting is played at that time.

Use Case Six - Tone Direction Interaction Use Cases

1. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE

- If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE

2. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
3. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
4. Service Parameter = PLAYTONE_NOLOCAL_OR_REMOTE; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
5. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
6. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_LOCALONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_NOLOCAL_OR_REMOTE
7. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
8. Service Parameter = PLAYTONE_LOCALONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
9. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_NOLOCAL_OR_REMOTE
   - If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
   - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
10. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_LOCALONLY
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
11. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_REMOTEONLY

- If Mode is Silent, then Effective Tone Direction = PLAYTONE_REMOTEONLY
- If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY

12. Service Parameter = PLAYTONE_REMOTEONLY; API Request (startMonitor/updateMonitorType) = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY
13. Service Parameter = PLAYTONE_BOTHLOCALANDREMOTE; API Request (startMonitor/updateMonitorType) = Any tone
    - If Mode is Silent, then Effective Tone Direction = PLAYTONE_BOTHLOCALANDREMOTE
    - If Mode is Whisper, then Effective Tone Direction = PLAYTONE_REMOTEONLY

Use Case Seven

Supervisor B is a shared line and supervisor updates monitorType from silent to whisper. Address B is a shared line configured on terminal T1(initiator) and T2.

Whisper monitor: A is monitor target, B(T1) is monitor initiator. X calls A, A answers the call GC1 (ci1). B(T1) calls start monitor using GC2(mode = silent). Application has call observer on all A, B(T1) and B(T2_. Application has monitoring capability enabled. App updates the monitor type to Whisper and later drops monitoring call to stop monitoring.

Action	Events	Call information
A answers GC1	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called:A

Action	Events	Call information
B calls start monitor using GC2 giving CI1, A and TermA from GC1 and mode as Silent	GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv B(T1) GC2: TermConnActiveEv B(T1) GC2: TermConnCreatedEv for B(T2) GC2: TermConnPassiveEv for B(T2) GC2: CallCtlTermConnTalkingEv B(T1) B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv B(T1) getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv B(T1) Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.SILENT_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv TAgetMonitorType() = CiscoCall.SILENT_MONITOR GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:B, device name: B(T1) CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.SILENT_MONITOR	GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
The application calls updateMonitorType() API on TermConn of B with mode as Whisper	GC2: CiscoTermConnMonitorUpdatedEv TA GC2: CiscoTermConnMonitorUpdatedEv B(T1) getMonitorType() = CiscoCall.WHISPER_MONITOR getTransactionID() = X GC2: CiscoRTPOutputStartedEv B(T1)	
B calls drop on GC2 to stop monitoring	GC2: CallCtlTermConnDroppedEv BT(1) GC2: TermConnDroppedEv B(T1) GC2: CallCtlTermConnDroppedEv BT(2) GC2: TermConnDroppedEv B(T2) GC2: ConnDisconnectedEv B GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	

Use Case Eight

Agent is shared line. After monitoring Agent holds and its shared line resumes.

A(T1) is monitor target and shares a line with Terminal T2 , B is monitor initiator. X calls A, A answers the call GC1 (ci1). B calls start monitor using GC2. Application has call observer on all A(T1), A(T2) and B. Application has monitoring capability enabled.

Action	Events	Call information
A(T1) answers GC1B calls start monitor using GC2 giving CI1, A and TermA(T1) in GC1 and mode as Whisper		GC1: Calling: X Called: A LRP: null Current calling: X Current called:A GC2: Calling: B Called: A LRP: null Current calling: B Current called:A

Action	Events	Call information
	CallActiveEv for callID = GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X GC1:CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv T1 CiscoRTPInputStartedEv T1 GC2:CallActive Cause: CAUSE_NORMAL GC2: ConnCreatedEv for B Cause: CAUSE_NORMAL GC2: CallCtlConnInitiatedEv GC2: TermConnCreatedEv TB GC2: TermConnActiveEv TB GC2: CallCtlTermConnTalkingEv TB B Cause: CAUSE_NORMAL GC2: CallCtlConnDialingEv for B GC2: CallCtlConnEstablishedEv for B Cause: CAUSE_NORMAL GC2: ConnCreatedEv A , here A.getType() = MONITORING_TARGET GC2: ConnInProgressEv A GC2: CallCtlConnOfferedEv A, GC2: ConnAlertingEv A GC2: CallCtlConnAlertingEv A GC2: ConnConnectedEv A GC2: CallCtlConnEstablishedEv A GC2: CiscoRTPInputStartedEv TB GC2: CiscoRTPOutputStartedEv TB getCiscoFeatureReason() = CiscoFeatureReason.REASON_CALL_MONITORING GC2:CiscoTermConnMonitorTargetInfoEv TB Cause: CAUSE_NORMAL address:A, here A.getType() = MONITORING_TARGET, terminal name: TA, rtphandle = CI1 CiscoMonitorTagetInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR [Note: Above connection is not created on Observered address A, rather an another Address object of type MONITORING_TARGET.] GC1: CiscoTermConnMonitoringStartEv T1(A) getMonitorType() = CiscoCall.WHISPER_MONITOR	

Action	Events	Call information
	GC1: CiscoTermConnMonitorInitiatorInfoEv T1(A) Cause: CAUSE_NORMAL address:B, device name: TB CiscoMonitorInitiatorInfo.getMonitorType() = CiscoCall.WHISPER_MONITOR	
A(T1) puts the call on hold	GC1: CallCtlTermConnHeldEv A(T1) GC1: CallCtlTermConnActiveEv A(T2) GC1: CallCtlTermConnHeldEv A(T2) GC2:CiscoRTPInputStoppedEv TB GC2: CiscoRTPOutputStoppedEv TB GC1: CiscoRTPOutputStoppedEv A(T1) GC1: CiscoRTPInputStoppedEv A(T1)	
A(T2) resumes the call	GC1: CallCtlTermConnTalkingEv A(T2) GC1: CallCtlTermConnPassiveEv A(T1) GC1: CiscoRTPOutputStartedEv A(T2) GC1: CiscoRTPInputStartedEv A(T2) GC2: CallCtlTermConnDroppedEv TB GC2: TermConnDroppedEv TB GC2: ConnDisconnectedEvB GC2: CallCtlConnDisconnectedEv B GC2: ConnDisconnectedEv A GC2: CallCtlConnDisconnectedEv A GC2: CallInvalidEv GC1: CiscoTermConnMonitorEndEv TA	