Message Sequence Charts

This appendix contains message sequences or call scenarios and illustrates a subset of these scenarios that are supported by the Cisco Unified TSP. Be aware that the event order is not guaranteed in all cases and can vary depending on the scenario and the event.

This appendix contains the following sections:

Abbreviations

The following list gives abbreviations that are used in the CTI events that are shown in each scenario:

- NP—Not Present
- LR—LastRedirectingParty
- CH—CtiCallHandle
- GCH—CtiGlobalCallHandle
- RIU—RemoteInUse flag
- DH—DeviceHandle

3XX

Application monitors B.

Table 1: 3XX

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
A calls external phone that is running SIP, which has CFDUNC set to B		TSPI: LINE_APPNEWCALL Reason = LINECALL REASON_REDIRECT	

Agent Greeting

Configuration

Customer Phone—IP Phone A with DN 1001.

Agent Phone—IP Phone B with DN 1002.

Agent Phone—IP Phone C with DN 1002 (shared line)

Supervisor Phone—IP Phone D with DN 1003.

IVR1—with DN 5555

IVR2—with DN 6666

Procedure

Application monitoring all lines on all devices.

New extension is negotiated when application opens lines.

SRTP is also supported at IVR side, can be variation of following use cases.

Table 2: StartSendMediaToBIB Success Case

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 with 5555 and CgpnToIVR (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB by feature IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit is set Media event sent to application (StartTransmissionEvent)

Action	Events, requests and responses
IVR1 drops call after agent greeting completes	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event At 5555: Call goes IDLE

Table 3: StopSendMediaToBIB Success Case

Action	Events, requests and responses
Agent playing is in progress…	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001 At 5555: CONNECTED Calling = 5555 Called = 5555 Connected =
Application issues CCiscoLineDevSpecificStopSendMediaToBIBRequest on 1002	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event At 5555: Call goes IDLE StopTransmissionEvent

Table 4: StartSendMediaToBIB Failure While Monitoring in Progress at Agent Side

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application issues CCiscoLineDevSpecificStartCallMonitoring on 1003 to monitor active call on 1002	At 1003: CCiscoLineDevSpecificStartCallMonitoring request successful, monitoring is in session
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002	At 1002: LINE_REPLY returns with LINEERR_RESOURCEUNAVAIL

Table 5: StartSendMediaToBIB Followed by Monitoring Request

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Application issues CCiscoLineDevSpecificStartCallMonitoring on 1003 to monitor active call on 1002	At 1003: LINE_REPLY returns with LINEERR_RESOURCEUNAVAIL

Table 6: StartSendMediaToBIB While Recording Is in Session

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application sends CCiscoLineDevSpecificStartCallRecording to 1002	At 1002: CCiscoLineDevSpecificStartCallRecording will be successful and recording is in session
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)

Action	Events, requests and responses
IVR1 drops call after agent greeting completes	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event At 5555: Call goes IDLE

Table 7: StartSendMediaToBIB Followed by Recording Request

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Application sends CCiscoLineDevSpecificStartCallRecording to 1002	At 1002: CCiscoLineDevSpecificStartCallRecording will be successful and recording is in session

Table 8: StartSendMediaToBIB Failure While Barge in Session

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Phone C (1002) barges in	At 1002 (device C) Barge call is created.
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (B)	At 1002 (B): LINE_REPLY with LINEERR_RESOURCEUNAVAIL

Table 9: StartSendMediaToBIB Followed by Barge From Shared Line

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Phone C (1002 shared line) try to barge in	Barge will fail on phone C

Table 10: This Behavior Is Also Seen During Consult Operation. Agent Holds Call While Agent Greeting Is Being Played

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
1002 put call on hold	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event Call will go on hold With StopReception and StopTransmission event At 5555: Call goes IDLE
1002 Unhold scenario	At 1002: Call will go CONNECTED with StartTransmission and StartReception.

Table 11: Agent Redirects Call While Agent Greeting Is Being Played

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)

Action	Events, requests and responses
Application redirects call on 1002 to 1003	At 1003: New call from 1002 At 1002: Call goes IDLE No MEDIA_TO_BIB_ENDED event At 5555: Call goes IDLE

Table 12: IVR1 Redirects Call to IVR2

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Application redirect call on IVR1 to IVR2 IVR2 answers and plays second agent greeting	At 5555: Call goes IDLE At 6666: Calling = Called = 6666 Connected = Redirecting = 5555 Redirection = 6666 CallAttributeBitMask = BIBCall (StartTransmissionEvent)
IVR2 drops call after agent greeting completes	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event At 6666: Call goes IDLE

Table 13: Application-2 Opened Line After Agent Greeting Is in Playing

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application-1 issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 with 5555 and CgpnToIVR (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB by feature IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Application-2 opens agent line from another client	At 1002 (from application-2): CallAttributeBitMask SendMediaToBIB will be set to indicate agent greeting is playing on the agent line.
Application 2 opens IVR line	CallAttributeBitMask = BIBCall

Table 14: Start Agent Greeting After Conference Is Setup

Action	Events, requests and responses
Make call from 1001 to 1002, 1002 answers, 1002 sets up conference to 1003, 1003 answers, and 1002 completes	At 1001: CONNECTED CONFERENCED Calling = 1001, Called = 1002, Connected = 1002 CONFERENCED Calling = 1001, Called = 1003, Connected = 1003At 1002: CONNECTED CONFERENCED Calling = 1001, Called = 1002, Connected = 1001 CONFERENCED Calling = 1002, Called = 1003, Connected = 1003 At 1003: CONNECTED CONFERENCED Calling = 1002, Called = 1003, Connected = 1002 CONFERENCED Calling = 1003, Called = 1001, Connected = 1001

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 with 5555 and CgpnToIVR (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB by feature IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent) 1001 and 1002 also hears the agent greeting

Agent Zip Tone

The devices mentioned in the use cases below also apply to SIP TNP phones.

Configuration

SCCP phones: A (Customer/Remote), B (Agent/Local).

All Lines are Opened with Ext Version – 0x000B0000

Table 15: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – Remote

Action	Expected events
LineInitialize. LineOpen on A,B The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line A and B. A calls B;B answers the Call B issues LineDevSpecific (start PlayTone) request with Agent callid and ZIP Tone as input.	Zip Tone is played at A. LINE_DEVSPECIFIC Event with dwParam1 = SLDSMT_CALL_TONE_CHANGEDdwParam2 = CTONE_ZIP, dwParam3 = 0(local) is reported on A and alsoLINE_DEVSPECIFIC Event with dwParam1 = SLDSMT_CALL_TONE_CHANGEDdwParam2 = CTONE_ZIP, dwParam3 = 1(Remote) is reported on B.

Table 16: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – Local

Action	Expected events
LineInitialize. LineOpen on A,B The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line A and B. A calls B;B answers the Call B issues LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input.	Zip Tone is played at B. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local) is fired for B indicating Zip Tone has been played on B.

Table 17: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – BothLocalandRemote/NoLocalOrRemote

Action	Expected events
LineInitialize. LineOpen on A,B A calls B; B answers the Call B issues LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input	LineDevSpecific (start PlayTone) request fails with error LINEERR_OPERATIONUNAVAIL.

Table 18: Application Issues the Play Tone Request (with Unsupported Tone) When the Call Is Established Between Customer and Agent. PlayToneDirection – Local

Action	Expected events
LineInitialize. LineOpen on A,B A calls B; B answers the Call B issues LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input	LineDevSpecific (start PlayTone) request fails with error LINEERR_OPERATIONFAILED.

Application Issues the Play Tone Request on a CTI Port with PlayToneDirection -Local/Remote

Configuration

A (Customer/Remote) is SCCP Phone.

B (Agent/local) is a CTIport/Route Point

Table 19: Application Issues the Play Tone Request on a CTI Port with PlayToneDirection – Local/Remote

Action	Expected events
LineInitialize. LineOpen on A,B The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line A. A calls B;B answers the Call B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input, and direction as local. B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input, and direction as remote.	LineDevSpecific (start PlayTone) request fails with error LINEERR_OPERATIONUNAVAIL. Zip Tone is played at A. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local)) is fired for A indicating Zip Tone has been played on A And also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote) is fired for B

Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent (Shared Line). PlayToneDirection -Local

Configuration

SCCP phones: A (Customer/ Remote), B, B’ (Agent/Local)

Table 20: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent (Shared Line). PlayToneDirection – Local

Action	Expected events
LineInitialize. LineOpen on A, B, B' The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B and B'. A calls B;B and B' starts ringing; B answers the Call B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input. Variants: B' issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote. B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote. A issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote.	Zip Tone is played at B. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local)) is fired for B indicating Zip Tone has been played on B. There is no Zip Tone played at B'and no Zip tone notification on B'. The LineDevSpecific (start PlayTone) request fails with Error LINEERR_OPERATIONFAILED Zip Tone is played at A. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local))) will be fired for A also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote) will be fired for B. There is no Zip Tone played at B'and no Zip tone notification on B'. Zip Tone is played at B and B'. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local))) is fired for B and B' also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote) is fired for A.

Table 21: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent (Intercom Line). PlayToneDirection – Local

Action	Expected events
LineInitialize. Phone A have 2 lines: Line1 is a normal line with X, Line2 is a intercom line (B), SpeedDial DN = D Phone B have 2 lines: Line1 is a normal line with Y, Line2 is a intercom line (D) LineOpen on B,D The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B, D B calls D; D starts ringing; D answers the Call D issues the LineDevSpecific (start PlayTone) request with agent(D) callid and ZIP Tone as input. Variant 1: D issues the LineDevSpecific (start PlayTone) request with agent(D) callid and ZIP Tone as input, and direction as remote.	The LineDevSpecific (start PlayTone) request fails with error LINEERR_OPERATIONUNAVAIL. The LineDevSpecific (start PlayTone) request fails with error LINEERR_OPERATIONUNAVAIL.

Conference Scenario: PlayToneDirection -local.

Configuration

A, B, and C are SCCP Phones.

Table 22: Conference Scenario. PlayToneDirection – Local

Action	Expected events
LineInitialize. LineOpen on A, B, and C The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B. A calls B; B answers the call; B sets up the conference with C; B completes the conference. B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input. Variant 1: B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input and direction as Remote	Zip Tone is played at B. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local)) is fired for B indicating Zip Tone has been played on B. The LineDevSpecific (start PlayTone) request will be Success. But there will be no Tone played on the Coneference members. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote)) is fired for B

Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent Agent Puts the Call on Hold. PlayToneDirection -Remote

Configuration

A and B are SCCP Phones.

Table 23: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent, Agent Puts the Call on Hold. PlayToneDirection – Remote

Action	Expected events
LineInitialize. LineOpen on A,B The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B. A calls B;B answers the Call; B puts the Call on hold A issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input.	Zip Tone is played at B. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote)) is fired for A also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local) is fired for B.

Announcement Call

Prerequisites

Pre-conditions to all announcement call use cases, unless specified otherwise:

- CTIRD (CTI Remote Device -Name: CTIRD-1)
  - Remote Destinations configured on CTIRD-1:
    - RD1-(Name: Mobile, Number: 914086271309)
  - Line-A (DN -1000) - Line-A configured on CTIRD-1 (shared line of Enterprise
  - DN -1000 configured on EP-1)
- EP-1 (Enterprise Phone - SCCP -IP Phone)
  - Line-A' -DN -1000 configured on EP-1
- Provider is opened (lineInitializeEx successfully executed)
- All relevant lines are opened with Extension version 0x000D0000 and in service

Persistent call has been created on A / RD-1.

Announcement with ID "WelcomeID" is defined on CUCM.

Table 24: Create Announcement Call

Action	TAPI Messages	TAPI Structure

Create Announcement Call:
LineMakeCall() on Line-A:
lpCallParams:
devSpecific =
Cisco_ CallParamsDevSpecific {
dwCallPriority = 0x00000000;
dwDevSpecificFlags = 0x00000004
(Cisco_ CALLPARAMS_ DEVSPECIFICFLAGS_ ANNOUNCEMENTCALL)
}
CallData = "WelcomeID"
	LINE_ CALLSTATE
	hDevice = hCall-2 dwParam1 = 0x40000002
	(CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + OFFERING)
	LINE_ CALLSTATE dwParam1 = 0x40000004
	(CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + ACCEPTED)	

	LINE_ CALLSTATE hDevice = hCall-2 dwParam1 = 0x40000100 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + CONNECTED) LINE_ CALLDEVSPECIFIC hDevice = hCall-2 dwParam1 = SLDSMT_ ANNOUNCEMENT_ STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) dwOrigin = OUTBOUND dwReason = DIRECT CallerID = 5000 CallerIDName = RD5000 CalledID = A ConnectedID = 5000 **In DevSpecific portion:** CallAttributeType = 0x00008000 (TSPCallAttribute_ AnnouncementCall)
	LINE_ CALLDEVSPECIFIC hDevice = hCall-2 dwParam1 = SLDSMT_ ANNOUNCEMENT_ ENDED dwParam2 = 0 dwParam3 = 0	
	LINE_ CALLSTATE dwParam1 = 0x40004000 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + DIS CONNECTED)	

Action	TAPI Messages	TAPI Structure
	LINE_ CALLSTATE dwParam1 = 0x40000001 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + IDLE)	

Persistent call has been created on A / RD-1.

Announcement with ID "WelcomeID" is defined on CUCM.

Table 25: Drop Announcement Call

Action	TAPI Messages	TAPI Structures

Create Announcement Call:
LineMakeCall() on Line-A:
lpCallParams:
devSpecific =
Cisco_ CallParamsDevSpecific {
dwCallPriority = 0x00000000;
dwDevSpecificFlags = 0x00000004
(Cisco_ CALLPARAMS_ DEVSPECIFICFLAGS_ ANNOUNCEMENTCALL)
}
CallData = "WelcomeID"
	LINE_ CALLSTATE
	hDevice = hCall-2 dwParam1 = 0x40000002
	(CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + OFFERING)
	LINE_ CALLSTATE dwParam1 = 0x40000004
	(CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + ACCEPTED)	

	LINE_ CALLSTATE hDevice = hCall-2 dwParam1 = 0x40000100 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + CONNECTED) LINE_ CALLDEVSPECIFIC hDevice = hCall-2 dwParam1 = SLDSMT_ ANNOUNCEMENT_ STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) dwOrigin = OUTBOUND dwReason = DIRECT CallerID = 5000 CallerIDName = RD5000 CalledID = A ConnectedID = 5000 **In DevSpecific portion:** CallAttributeType = 0x00008000 (TSPCallAttribute_ AnnouncementCall)

Action	TAPI Messages	TAPI Structures
Drop AnnouncementtCall: (while announcement being played) LineDrop() on Line-A:	LINE_ CALLDEVSPECIFIC hDevice = hCall-2 dwParam1 = SLDSMT_ ANNOUNCEMENT_ ENDED dwParam2 = 0 dwParam3 = 0 LINE_ CALLSTATE dwParam1 = 0x40004000 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + DIS CONNECTED) LINE_ CALLSTATE dwParam1 = 0x40000001 (CLDSMT_ ANNOUNCEMENT_ CALL_ STATE + IDLE)	

Precondition: No Persistent call on CTIRD-1

Table 26: Negative -Create Announcement Call Failed / No Persistent Call

Action	TAPI Messages	TAPI Structures

Create Announcement Call:
LineMakeCall() on Line-A:
lpCallParams:
devSpecific =
Cisco_ CallParamsDevSpecific {
dwCallPriority = 0x00000000;
dwDevSpecificFlags = 0x00000004
(Cisco_ CALLPARAMS_ DEVSPECIFICFLAGS_ ANNOUNCEMENTCALL)
}
CallData = "WelcomeID"
	LINE_ REPLY
	LINEERR_ NO_ PERSISTENT_ CALL_ EXISTS (0xC0000021)	

Blind Transfer

The following table describes the message sequences for Blind Transfer when A calls B, B answers, and A and B are connected.

Table 27: Message Sequences for Blind Transfer

Action	CTI messages	TAPI messages	TAPI structures
Party B does a lineBlindTranfser() to blind transfer call from party A to party C	Party A		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CalledChanged = True, Called = C, OriginalCalled = B, LR = B, Cause = BlindTransfer	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID, REDIRECTINGID, REDIRECTIONID	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = NP dwRedirectionID = NP
	Party B		
	CallStateChangedEvent, CH = C2, State = Idle, Reason = Direct, Calling = A, Called = B, OriginalCalled = B, LR = NULL	TSPI: LINE_CALLSTATE |hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = NULL dwRedirectionID = NULL
	Party C		
	NewCallEvent, CH = C3, origin = Internal_Inbound, Reason = BlindTransfer, Calling = A, Called = C, OriginalCalled = B, LR = B	TSPI: LINE_APPNEWCALL hDevice = C dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = TRANSFER dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C
	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangeEvent, CH = C1, State = Ringback, Reason = Direct, Calling = A, Called = C, OriginalCalled = B, LR = B	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C
	Party C		
	CallStateChangedEvent, CH = C3, State = Offering, Reason = BlindTransfer, Calling = A, Called = C, OriginalCalled = B, LR = B	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = OFFERING dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C

Call Control Discovery

Basic Call Initiated From TAPI From Phone A on Cluster 1 to Phone B on Cluster2

Configuration

SCCP phone A(1900) are registered to cluster A

Phones A are associated with the end-user cluster1

SCCP phone B(1000) registered to cluster B

Phones B are associated with the end-user cluster2

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network. TAPI is observing A.

Procedure

Application monitors A

Application sends a lineMakeCall at A to call B

Action	CTI messages	TAPI messages
A dials 1000, this call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “”
SIP trunk rejects this call due to no more bandwidth available	A receives CallStateChangeEvent (PROCEEDING)	LineA: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT CallerID = A / CalledID = 1000 / ConnectedID = / RedirectingID = / RedirectionID =
CCD Requesting feature will start PSTN failover by directing this caller to 1000’s PSTN failover number, that is, 14089721000. Call is sent out to a PSTN GW		

Action	CTI messages	TAPI messages
CCD Requesting feature will start PSTN failover by directing this caller to 1000's PSTN failover number, that is, 14089721000. Call is sent out to a PSTN GW	A receives CPIC and CallStateChangeEvent (Ringback/connected) Provide TSPI_LinegetcallInfo on A connected with B	A:CPIC event received on party A LineA: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) CallReason = LINECALLREASON_DIRECT LINECALLINFO.dwCallID = 0x00400BBA LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1900(A) LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1000: LINECALLINFO.dwCalledIDName = CCD Pattern LINECALLINFO.dwConnectedID = 1000(B) LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1000 LINECALLINFO.dwRedirectingIDName = CCD Pattern ExtendCallReason = CtiReasonSAF_CCD_PSTNFailover(2B)

Basic Call Initiated From TAPI From Phone A on Cluster 1 to Phone B on Cluster2 with PSTN Failover Rule Not Set

Configuration

SCCP phone A are registered to cluster A.

Phones A are associated with the end-user "cluster1".

SCCP phone B(1000) registered to cluster B.

Phones B are associated with the end-user "cluster2".

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network is not set.

Procedure

Application monitors A.

Application sends a lineMakeCall at A to call B.

Action	CTI messages	TAPI messages
A dials 1000, this call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “”
SIP trunk rejects this call due to lack of bandwidth	A receives CallStateChangeEvent (PROCEEDING)	A:A receives CPIC event LineA: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT CallerID = A / CalledID = 1000 / ConnectedID = / RedirectingID = / RedirectionID =
	A receives CallStateChangeEvent (disconnected)	LineA: LINE_CALLSTATE (LINECALLSTATE_Disconnected) EVENT = LINE_CALLSTATE = 2 m_lpfnEventProc = 0xXXX m_htLine = 0x000XXXX htCall = 0x000XXX

Action	CTI messages	TAPI messages
	Provide TSPI_linegetcallinfo on the Disconnected call	dwParam1 = 0x00004000(LINECALLSTATE_DISCONNECTED) dwParam2 = 0x00200000(LINEDISCONNECTMODE_SAFCCD) dwParam3 = 0x00000004 LINECALLINFO.dwCallID = 0x00400BCF LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1900 LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 10XX: LINECALLINFO.dwCalledIDName = CCD Pattern LINECALLINFO.dwConnectedID = LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000: LINECALLINFO.dwRedirectionIDName = CCD Pattern LINECALLINFO.dwRedirectingID = 1000: LINECALLINFO.dwRedirectingIDName = CCD Pattern ExtendCallReason = CtiReasonSAF_CCD_PSTNFailover

Basic Call Initiated From TAPI From Phone A(1900) and B(1901) on Cluster 1 B Redirects to Phone C(1000) on Cluster2 with PSTN Failover Rule Set

Configuration

SCCP phone A and B are registered to cluster A.

Phones A and B are associated with the end-user cluster1.

SCCP phone C(1000) registered to cluster B.

Phones C are associated with the end-user cluster2.

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network.

Procedure

Application monitors A and B.

Application sends a lineMakeCall at A to call B

Table 28: Basic Call Initiated From TAPI From Phone A(1900) and B(1901) on Cluster 1, B Redirects to Phone C(1000) on Cluster2 with PSTN Failover Rule Set

Action	CTI messages	TAPI messages
A dials B	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing/Proceeding /ringback/connected). B receives NewCallEvent and CallStateChangeEvent (offering/ringing/ connected).	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING, LINECALLSTATE_CONNECTED,) CallerID = A / CalledID = B B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_RINGING, LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B
B setsupconference, consult call to C(1000), this call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk SIP trunk rejects this call due to no more bandwidth available CCD Requesting feature will start PSTN failover by directing this caller to 1000's PSTN failover number, i.e. 14089721000. Call is sent out to a PSTN GW TSPI_linegetcallinfo on the consult call between B and C. B completes conference.		B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) B: receives CPIC event LineB: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) ExtendCallReason = CtiReasonSAF_CCD_PSTNFailover A, B and C are in conference.

Basic Call Initiated From TAPI From Phone A and B on Cluster 1 B Transfers to Phone C(1000) on Cluster 2 with PSTN Failover Rule

Configuration

SCCP phone A and B are registered to cluster A.

Phones A(1900) and B(1901) are associated with the end-user cluster1.

SCCP phone C(1000) registered to cluster B.

Phones C are associated with the end-user cluster2.

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network.

Procedure

Application monitors A and B.

Application sends a lineMakeCall at A to call B.

Table 29: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Transfers to Phone C(1000) on Cluster 2 with PSTN Failover Rule

Action	CTI messages	TAPI messages
A(1900) dials B(1901)	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing/Proceeding /ringback/connected). B receives NewCallEvent and CallStateChangeEvent (offering/ringing/ connected)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING, LINECALLSTATE_CONNECTED,) B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_RINGING, LINECALLSTATE_CONNECTED)

Action	CTI messages	TAPI messages
B(1901) setups transfer to C(1000) This call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk SIP trunk rejects this call due to no more bandwidth available CCD Requesting feature will start PSTN failover by directing this caller to 1000's PSTN failover number, i.e. 14089721000. Call is sent out to a PSTN GW. TSPI_linegetcallinfo on Consult call on B with C. B completes transfer		B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) LINECALLINFO.dwCallID = 0x00400BBA LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1901(B) LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1000: LINECALLINFO.dwCalledIDName = CCD Pattern LINECALLINFO.dwConnectedID = 1000(C) LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1000 LINECALLINFO.dwRedirectingIDName = CCD Pattern Extendedcallreason = CtiReasonSAF_CCD_PSTNFailover B: LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED) ExtendCallReason = CtiReasonTransferredCall

Call Initiated From TAPI From Phone A and B on Cluster 1 B Sets Up Conference to Phone C(1000) on Cluster 2 with PSTN Failover Rule

Configuration

SCCP phone A and B are registered to cluster A

Phones A(1900) and B(1901) are associated with the end-user cluster1

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 30: Call Initiated From TAPI From Phone A and B on Cluster 1, B Sets Up Conference to Phone C(1000) on Cluster 2 with PSTN Failover Rule

Action	CTI messages	TAPI messages
A dials B	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing/Proceeding /ringback/connected) B receives NewCallEvent and CallStateChangeEvent (offering/ringing/ connected)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING, LINECALLSTATE_CONNECTED,) CallerID = A / CalledID = B B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_RINGING, LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B

Action	CTI messages	TAPI messages
B setsupconference, consult call to C(1000), this call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk SIP trunk rejects this call due to no more bandwidth available CCD Requesting feature will start PSTN failover by directing this caller to 1000's PSTN failover number, that is, 14089721000. Call is sent out to a PSTN GW TSPI_linegetcallinfo on the consult call between B and C B completes conference	B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) B: receives CPIC event LineB: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) ExtendCallReason = CtiReasonSAF_CCD_PSTNFailover A, B and C are in conference	

Basic Call Initiated From TAPI From Phone A on Cluster 1 to Phone B on Cluster 2 Over SAF Trunk

Configuration

SCCP phone A(1900) are registered to cluster A

Phones A are associated with the end-user cluster1

SCCP phone B(1000) registered to cluster B

Phones B are associated with the end-user cluster2

CUCM learns a pattern 10XX, no PSTN failover rule as SAF network has unlimited Bandwidth, TAPI is observing A

Procedure

Application monitors A

Application sends a lineMakeCall at A to call B

Table 31: Basic Call Initiated From TAPI From Phone A on Cluster 1 to Phone B on Cluster 2 Over SAF Trunk

Action	CTI messages	TAPI messages
A dials 1000	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = ""

Action	CTI messages	TAPI messages
	A receives CallStateChangeEvent (PROCEEDING)	LineA: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT CallerID = A / CalledID = 1000 / ConnectedID = / RedirectingID = / RedirectionID =
	A receives CallStateChangeEvent (Ringback/connected)	A:CPIC event received on party A LineA: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) CallReason = LINECALLREASON_DIRECT CallerID = A / CalledID = 1000 / ConnectedID = 1000 / RedirectingID = 1000 / RedirectionID = 1000 LINECALLINFO.dwCallID = 0x00400FB1 LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1900 LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1000: LINECALLINFO.dwCalledIDName = CCD Pattern LINECALLINFO.dwConnectedID = 1000 LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1000: LINECALLINFO.dwRedirectingIDName = CCD Pattern

Basic Call Initiated From TAPI From Phone A and B on Cluster 1 B Redirects to Phone C(1000) on Cluster 2 Over SAF Trunk

Configuration

SCCP phone A and B are registered to cluster A

Phones A and B are associated with the end-user cluster1

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

CUCM learns a pattern 10XX, from SAF network as unlimited Bandwidth

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 32: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Redirects to Phone C(1000) on Cluster 2 Over SAF Trunk

Action	CTI messages	TAPI messages
A dials B	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing/Proceeding /ringback/connected) B receives NewCallEvent and CallStateChangeEvent (offering/ringing/ connected)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING, LINECALLSTATE_CONNECTED,) B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_RINGING, LINECALLSTATE_CONNECTED)

Action	CTI messages	TAPI messages
B redirects call to 1000 over ICT trunk TSPI_linegetcallinfo on A	A receives CallStateChangeEvent (Connected)	B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED) ExtendCallReason = CtiReasonRedirect A:CPIC event received on A LineA: LINE_CALLSTATE (LINECALLSTATE_RINGBACK) LineA: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT LINECALLINFO.dwCallID = 0x00400FB2 LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1900 LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1901 LINECALLINFO.dwCalledIDName = LINECALLINFO.dwConnectedID = 1000 LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1901 LINECALLINFO.dwRedirectingIDName = ExtendCallReason = CtiReasonRedirect

Basic Call Initiated From TAPI From Phone A and B on Cluster 1 B Transfers to Phone C(1000) on Cluster 2 Over SAF Trunk

Configuration

SCCP phone A and B are registered to cluster A

Phones A and B are associated with the end-user cluster1

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

CUCM learns a pattern 10XX, plus PSTN failover rule as 0:1408972 from SAF network, SAF network has unlimited bandwidth.

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 33: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Transfers to Phone C(1000) on Cluster 2 Over SAF Trunk

Action	CTI messages	TAPI messages
A calls B	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing/Proceeding /ringback/connected) B receives NewCallEvent and CallStateChangeEvent (offering/ringing/ connected)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING, LINECALLSTATE_CONNECTED,) B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_RINGING, LINECALLSTATE_CONNECTED)
B setup transfers to C(1000), through the ICT(SAF) trunk Complete transfer on B TSPI_linegetcallinfo on disconnected call on B	B: receives CPIC event	B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING/Proceeding) LineB: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) CallReason = LINECALLREASON_DIRECT B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED) ExtendCallReason = CtiReasonTransferredCall

Action	CTI messages	TAPI messages
TSPI_linegetcallinfo on A	A receives CallStateChangeEvent (Connected)	A: LineA: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT LINECALLINFO.dwCallID = 0x00400FB4 LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1000 LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1901 LINECALLINFO.dwCalledIDName = LINECALLINFO.dwConnectedID = 1000 LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1900 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1901 LINECALLINFO.dwRedirectingIDName = ExtendCallReason = CtiReasonTransferredCall

CallFwdAll Notification

This section describes the CallFwdAll Notification usecases.

Application Pressed CFwdAll on TAPI Monitored Device

Application opens the line with new ExtVersion 0x000A0000. User presses CFwdAll softkey on A when device is in on-hook condition.

Action	CTI events	Expected results
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000		
User presses CFwdAll softkey	NewCallEvent received for A	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask 0x00000040

TAPI Monitored Device Goes Off Hook

Application opens the line with new ExtVersion 0x000A0000. Device goes off hook.

Action	CTI events	Expected results
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000		
A goes off-hook	NewCallEvent received for A	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000000

Application Monitors Off Hook Device

Device goes off hook. Application does a LineInitialize and opens line A with new ExtVersion 0x000A0000

Action	CTI events	Expected results
Device goes offhook		
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000	ExistingCallEvent received at A	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallType 00000000

Application Monitors Device After User Presses CFwdAll

User presses CFwdAll softkey on the device. Application does a LineInitialize and opens line A with new ExtVersion 0x000A0000.

Action	CTI events	Expected results
User presses CFwdAll softkey on the device		

Action	CTI events	Expected results
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000	ExistingCallEvent received for A	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000040

User Presses CFwdAll Softkey After Device Is Off Hook

TAPI application does a LineInitialize and opens line A with new ExtVersion 0x000A0000. Device goes off hook and user presses CFwdAll softkey.

Action	CTI events	Expected results
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000	ExistingCallEvent received for A	
A goes off-hook User presses CFwdAll softkey	NewCallEvent received for A	LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000040
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000000

User Presses CFwdAll Softkey on a Multiline Device

TAPI application does LineInitialize and opens all lines-A1 and A2 for the device with new ExtVersion 0x000A0000. User presses the CFwdAll softkey.

Action	CTI events	Expected results
LineInitialize LineOpen on A1, LineOPen on A2 with new ExtVesrion 0x000A0000		
User presses CFwdAll softkey	NewCallEvent received for A1	
LineGetCallInfo on A1		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000040

User Presses CFwdAll on a Multiline Device by Selecting a Line

TAPI application does a LineInitialize and opens all lines-A1 and A2 for the device with new ExtVersion 0x000A0000. User selects line A2 and presses CFwdAll softkey.

Action	CTI events	Expected results
LineInitialize LineOpen on A1, LineOPen on A2 with new ExtVesrion 0x000A0000		
User selects line A2 and presses CFwdAll softkey	NewCallEvent received for A1	
LineGetCallInfo on A2		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000000

Shared Line Scenario on Pressing CFwdAll Softkey

TAPI application does a LineInitialize and opens a shared line A with new ExtVersion 0x000A0000 on devices P and Q. User presses CFwdAll softkey on device P.

Action	CTI events	Expected results
LineInitialize LineOpen on A LineOpen on A' with new ExtVesrion 0x000A0000		
On device P, user presses 'CFwdAll' softkey	NewCallEvent received at A NewCallEvent received at A' for RIU call	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000040
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000000

Cancellation of CFwdAll

TAPI application does a LineInitialize and open line A with new ExtVersion 0x000A0000. User sets CFwdAll for line A by pressing CFwdAll softkey followed by CallFwdAll destination number.

Later, user presses 'CFwdAll' softkey again to cancel CFwdAll setting.

Action	CTI events	Expected results
LineInitialize LineOpen on A with new ExtVesrion 0x000A0000		
User presses CFwdAll and enters FwdAll destination	NewCallEvent received for A	

Action	CTI events	Expected results
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000040
User again presses ‘CFwdAll’ softkey	NewCallEvent received for A	
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain CallAttributeBitMask : 0x00000080

Calling Party IP Address

Basic Call

TAPI application monitors party B

Party A represents an IP phone

A calls B

IP Address of A is available to TAPI application that is monitoring party B

Consultation Transfer

TAPI application monitors party C

Party B represents an IP phone

A talks to B

B initiates a consultation transfer call to C

IP Address of B is available to TAPI application that is monitoring party C.

B Completes the transfer

Calling IP address of A is not available to TAPI application that is monitoring party C (not a supported scenario).

Consultation Conference

TAPI application monitors party C

Party B represents an IP phone

A talks to B

B initiates a consultation conference call to C

IP Address of B is available to TAPI application that is monitoring party C.

B Completes the conference

Calling IP address of A and B is not available to TAPI application that is monitoring party C (not a supported scenario)

Redirect

TAPI application monitors party B and party C

Party A represents an IP phone

A calls B

IP Address of A is available to TAPI application that is monitoring party B

Party A redirects B to party C

Calling IP address is not available to TAPI application that is monitoring party B (not a supported scenario)

Calling IP address B is available to TAPI application that is monitoring party C

Calling Party Normalization

Incoming Call From PSTN to End Point

Action	CTI messages	TAPI messages	TAPI structures
A Call gets offered from a PSTN number 5551212/<SUBSCRIBER> through a San Jose gateway to a CCM end point 2000	CallStateChangedEvent, UnModified Calling Party = 5551212, UnModified Called Party = 2000, UnModified Original Called Party = 2000, Modified Calling Party = 5551212, Modified Called Party = 2000, Modified Original Called Party = 2000, Globalized Calling party = +14085551212, Calling Party Number Type = SUBSCRIBER, Called Party Number Type = UNKNOWN, Original Called Party Number Type, = UNKNOWN State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 5551212, Displayed Called Party = 2000, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +14085551212, Calling Party Number Type = SUBSCRIBER, Called Party Number Type = UNKNOWN, Redirection Party Number Type = , Redirecting Party Number Type =

Incoming Call From National PSTN to CTI-Observed End Point

Action	CTI messages	TAPI messages	TAPI structures
A Call gets offered from a Dallas PSTN number 5551212/<NATIONAL> through a San Jose gateway to a CCM end point 2000	CallStateChangedEvent, UnModified Calling Party = 9725551212, UnModified Called Party = 2000, UnModified Original Called Party = 2000, Modified Calling Party = 9725551212, Modified Called Party = 2000, Modified Original Called Party = 2000, Globalized Calling party = +19725551212, Calling Party Number Type = NATIONAL, Called Party Number Type = UNKNOWN, Original Called Party Number Type, = UNKNOWN State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 9725551212, Displayed Called Party = 2000, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +19725551212, Calling Party Number Type = NATIONAL, Called Party Number Type = UNKNOWN, Redirection Party Number Type = , Redirecting Party Number Type =

Incoming Call From International PSTN to CTI-Observed End Point

Action	CTI messages	TAPI messages	TAPI structures
A Call gets offered from a PSTN number in India 22221111/<INTERNATIONAL> through a San Jose gateway to a CCM end point 2000	CallStateChangedEvent, UnModified Calling Party = 011914422221111, UnModified Called Party = 2000, UnModified Original Called Party = 2000, Modified Calling Party = 011914422221111, Modified Called Party = 2000, Modified Original Called Party = 2000, Globalized Calling party = +914422221111, Calling Party Number Type = INTERNATIONAL, Called Party Number Type = UNKNOWN, Original Called Party Number Type, = UNKNOWN State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 011914422221111, Displayed Called Party = 2000, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +914422221111, Calling Party Number Type = INTERNATIONAL, Called Party Number Type = UNKNOWN, Redirection Party Number Type = , Redirecting Party Number Type =

Outgoing Call From CTI-Observed End Point to PSTN Number

Action	CTI messages	TAPI messages	TAPI structures
A Call gets initiated from a CCM end point 2000 through a San Jose gateway to a PSTN number 5551212/<NATIONAL>	CallStateChangedEvent, UnModified Calling Party = 2000, UnModified Called Party = 5551212, UnModified Original Called Party = 5551212, Modified Calling Party = 2000, Modified Called Party = 5551212, Modified Original Called Party = 5551212, Globalized Calling party = +14085551212, Calling Party Number Type = UNKNOWN, Called Party Number Type = SUBSCRIBER, Original Called Party Number Type, = SUBSCRIBER State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 2000, Displayed Called Party = 5551212, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +14085551212, Calling Party Number Type = UNKNOWN, Called Party Number Type = SUBSCRIBER, Redirection Party Number Type = , Redirecting Party Number Type =

Outgoing Call From CTI-Observed End Point to National PSTN Number

Action	CTI messages	TAPI messages	TAPI structures
A Call gets initiated from a CCM end point 2000 through a San Jose gateway to a Dallas PSTN number 9725551212/<NATIONAL>	CallStateChangedEvent, UnModified Calling Party = 2000, UnModified Called Party = 9725551212, UnModified Original Called Party = 9725551212, Modified Calling Party = 2000, Modified Called Party = 9725551212, Modified Original Called Party = 9725551212, Globalized Calling party = +19725551212, Calling Party Number Type = UNKNOWN, Called Party Number Type = NATIONAL, Original Called Party Number Type, = NATIONAL State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 2000, Displayed Called Party = 9725551212, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +19725551212, Calling Party Number Type = UNKNOWN, Called Party Number Type = NATIONAL, Redirection Party Number Type = , Redirecting Party Number Type =

Outgoing Call From CTI-Observed End Point to International PSTN Number

Action	CTI messages	TAPI messages	TAPI structures
A Call gets initiated from a CCM end point 2000 through a San Jose gateway to a PSTN number in India 914422221111/<INTERNATIONAL>	CallStateChangedEvent, UnModified Calling Party = 2000, UnModified Called Party = 011914422221111, UnModified Original Called Party = 011914422221111, Modified Calling Party = 2000, Modified Called Party = 011914422221111, Modified Original Called Party = 011914422221111, Globalized Calling party = +914422221111, Calling Party Number Type = UNKNOWN, Called Party Number Type = INTERNATIONAL, Original Called Party Number Type, = INTERNATIONAL State = Connected, Origin = OutBound, Reason = Direct	LINE_CALLSTATE = CONNECTED	LINECALLINFO Displayed Calling Party = 2000, Displayed Called Party = 011914422221111, Displayed Redirection Party = , Displayed Redirected Party = , Globalized Calling Party = +914422221111, Calling Party Number Type = UNKNOWN, Called Party Number Type = INTERNATIONAL, Redirection Party Number Type = , Redirecting Party Number Type =

Call PickUp

Registering CallPickUpGroup for Notification

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

Devices/Lines: 1000:P1,1001:P1.1002:P1,4000:P1 and 4001:P1

Pickup group P1:1111 is configured

P1:1000, P1:1001, P1:1002 are all in pickup group P1:1111

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application

Action	Expected events
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen for P1:1111	LineOpenSuccessful LineInService Event as well
LineInfo	DN and Partition information will be pickup Group DN and partition. LineName – "CtiCallPickupDevice" LineType -LINEDEVCAPSDEVSPECIFIC_PICKUPDN -0x00000004

UnRegistering CallPickUpGroup for Notification

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen for P1:1111	Line Open Successful
Application sends CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification on new line opened for PickUpGroup P1:1111	Line_Reply with success. LINE_REMOVE event will be sent to Application for P1:1111

Re-Registering CallPickUpGroup for Notification

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111

Action	Expected events
LineOpen for P1:1111	Line Open Successful
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with Error “LINEERR_OPERATIONUNAVAIL”
Variant : Test the Same with UnRegister	

Registering/UnRegistering CallPickUpGroup for Notification with Invalid Information

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with InValid DN or Partition	Line_Reply with Error Code “LINEERR_OPERATIONFAILED”
Variant : Test the Same with UnRegister	

CallPickUp After Enabling Auto Call Pickup Enabled

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen for P1:1111	Line Open Successful
P1:4000 calls P1:1002	LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111

Action	Expected events
LineGetCallInfo on new call on P1:1111	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin : Outbound dwCallReason : Direct Check for all fields of Calling and Called Information
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:1000	Events on P1:1000: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4000, Called = 1002, Connected = 4000, dwReason = Direct, dwOrigin = Internal. **Note** There is no notification at P1:1111 after the call has been pickup.
Varaint : P1:4000 calls P1:1002 and P1:4001 calls P1:1002 Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:1000	First incoming Call will be picked up (i.e call from 4000 will be picked up by 1000)

CallPickUp with Auto Call Pickup Enabled Disabled

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111

Action	Expected events
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:1000	Events on P1:1000: Call 1: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_IDLE **Note** First call will go IDLE state after Proceeding state. Call2: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_OFFERING Once the call is Answered LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4000, Called = 1002, Connected = 4000, dwReason = PickUp, dwOrigin = Outbound **Note** There is no notification at P1:1111 after the call has been pickup.
Varaint : Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:1002	CallPickup Request will be successful and the newcall will be created and the call will be in Offering state

CallPickUp with Multiple Calls Available

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful

Action	Expected events
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
P1:4001 calls P1:1001	Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
LineGetCallInfo on Call LineGetCallInfo on Call2	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4001 dwCalledID : 1001 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information

Action	Expected events
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:1000	Events on P1:1000: Call 3: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4000, Called = 1002, Connected = 4000, dwReason = Direct, dwOrigin = Internal **Note** There is no notification at P1:1111 after the call has been pickup.

CallPickupGroup Changed for a Device on AdminPage

Pickup group P1:9999 is configured

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on 1000:P1	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
Now from Admin page change the CallPickupGroup of 1000:P1 line to None or some other group P1:9999 LineGetDevCaps with Extension Version – 000A0000 on 1000:P1	Changed CallPickUp Group DN and Partition Information will be sent to application

CallPickUpGroup Partition or DN Information Updated

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on 1000:P1	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111

Action	Expected events
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information
Now From Admin Pages change the Partition or DN information of the Pickup Group	LINE_REMOVE for the line P1:1111
LineGetDevCaps with Extension Version – 000A0000 on 1000:P1	Changed CallPickUp Group DN and Partition Information will be sent to application

CallPickUpGroup Is Deleted

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
Now From Admin Pages Pickup Group 1111:P1 is deleted	LINE_REMOVE for the line P1:1111

Call Queuing

HP1 is a Huntpilot with the below configuration:

"Queue Calls" check box is selected.

“Display Line Group Member DN as Connected Party" check box is selected.

HP1: LG1

HP2: LG1

A, B (IP phones/CTI Ports)

Table 34: Basic Hunt List Call (HP1 Has at Least One Member Free)

Action	Expected events
App initiates call from A to HP1 and call is answered by LG1.	At A: LINE_CALLSTATE -RINGBACK At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = A HuntPilot = HP1

Table 35: Basic Hunt List Call. HP1 Has All Members Busy (LG1)

Action	Expected events
App initiates call from A to HP1 and call is Queued.	At A: LINE_CALLSTATE -RINGBACK At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1 Connected = HP1 HuntPilot =

Action	Expected events
Call on LG1 goes idle (LG1 is free). Queued call from A is de-queued and offered on LG1. LG1 Answers the call. Variance: Repeat and verify info when Display Line Group Member DN as Connected Party is enabled	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x2e(CallDeQueue) Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = A, Called = HP1 HuntPilot = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x2e(CallDeQueue) Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = A Called = HP1 HuntPilot = HP1 Connected = A Same as above

Table 36: Hunt List Call to HP1 When Queue Depth Is Reached. (Maximum Number of Callers Allowed in Queue = 2)

Action	Expected events
HP1 has 2 queued calls. App initiates call from A to HP1, call is disconnected	At A: LINE_CALLSTATE -DISCONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1

Action	Expected events
Variance: Destination When Queue is Full = B B Answers the call.	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At B: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1 HuntPilot = HP1 Connected = B At B: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A Called = HP1 Connected = A

Action	Expected events
Variance: Destination When Queue is Full = HP2 Call on LG1 of HP2 goes idle (LG1 is free). Queued call from A is de-queued and offered on LG1.	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 of HP2 HuntPilot = HP2 At LG1 of HP2: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A Called = HP1 Connected = A

Table 37: Hunt List Call to HP1 and Maximum Wait Time in Queue Is Met

Action	Expected events
HuntMember LG1 of HP1 is busy. App initiates call from A to HP1. Maximum wait time at queue is reached.	At A: LINE_CALLSTATE -RINGBACK At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = HP1 HuntPilot = At A: LINE_CALLSTATE -DISCONNECTED CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1

Action	Expected events
Variance: Destination When maximum wait time in Queue expires = B	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1 At B: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x2f(CallDeQueueTimerExpired) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1 HuntPilot = HP1 Connected = B At B: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x2f(CallDeQueueTimerExpired) Caller = A Called = HP1 Connected = A

Action	Expected events
Variance: Destination maximum wait time in Queue expires = HP2	

Action	Expected events
	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1 At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x2f(CallDeQueueTimerExpired) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 of HP2 HuntPilot = HP2 At LG1 of HP2: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x2f(CallDeQueueTimerExpired) Caller = A

Action	Expected events
	Called = HP1 Connected = A

Table 38: Hunt List Call to HP1 and No Agents Logged In or Registered

Action	Expected events
App initiates call from A to HP1. (None of the Huntmembers are registered or logged in). Destination When There Are No Agents Logged In or Registered = ' B' Call offered on B. B Answers the call.	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At B: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = B HuntPilot = At B: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A Called = HP1 Connected = A

Action	Expected events
App initiates call from A to HP1. (None of the Huntmembers are registered or logged in). Destination When There Are No Agents Logged In or Registered = 'HP2' Call offered on HP2. HP2 Answers the call.	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = B HuntPilot = HP2 At B: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A Called = HP1 Connected = A

Table 39: Basic Hunt List Call. A Calls B, and B Redirects/forwards/transfers the Call to HP1

Action	Expected events
App initiates call from A to B	At A: LINE_CALLSTATE -RINGBACK At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, Connected = B At B: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, Connected = A

Action	Expected events
The call on B is transferred to HP1 (Blind transfer).	

Action	Expected events
	At B: LINE_CALLSTATE -IDLE CallReason = x1(Direct) ExtendedCallReason = x7(BlindTransferCall) Caller = A Called = B, HuntPilot = Connected = HuntPilot = At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x100(LINECALLREASON_TRANSFER) ExtendedCallReason = x7(BlindTransferCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = HuntPilot = At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, HuntPilot = Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x100(LINECALLREASON_TRANSFER) Caller = A Called = HP1, HuntPilot = HP1

Action	Expected events
	Connected = A HuntPilot =

Action	Expected events
Variance: Call on B is redirected to HP1 LG1 Answers the call.	

Action	Expected events
	At B: LINE_CALLSTATE -IDLE CallReason = x1(Direct) ExtendedCallReason = x6(Redirect) Caller = A Called = B, HuntPilot = Connected = HuntPilot = At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x6(Redirect) Caller = A Called = B, HuntPilot = Connected = HuntPilot = At A: LIN_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, HuntPilot = Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x6(Redirect) Caller = A, Called = B

Action	Expected events
	HuntPilot = Connected = LG1 HuntPilot =

Action	Expected events
Variance: Call on B is forwarded to HP1 (Forward All) LG1 Answers the call.	

Action	Expected events
	At A: LINE_CALLSTATE -RING_BACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, HuntPilot = Connected = HuntPilot = At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x8(LINECALLREASON_FWDUNCOND) ExtendedCallReason = x5(ForwardAllCall) Caller = A Called = B, HuntPilot = Connected = HuntPilot = At A: LIN_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = B, HuntPilot = Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x8(LINECALLREASON_FWDUNCOND) ExtendedCallReason = x5(ForwardAllCall) Caller = A, Called = B

Action	Expected events
	Connected = LG1

Table 40: Basic Hunt List Call. HP1 Has All Members Busy (LG1), Queued Call on A Is Redirected

Action	Expected events
App initiates call from A to HP1 and call is Queued.	At A: LINE_CALLSTATE -RINGBACK At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x2d(CallQueue) Caller = A Called = HP1, HuntPilot = HP1 Connected = HP1 HuntPilot =

Action	Expected events
Queued Call on A is redirected to B. B Answers. Call on LG1 goes idle (LG1 is free). Queued call from B is de-queued and offered on LG1. LG1 Answers the call.	At B: LINE_CALLSTATE -CONNECTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x6(Redirect) Caller = HP1 Called = B, HuntPilot = Connected = HP1 HuntPilot = At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = B, Called = HP1 HuntPilot = HP1 At B: LINE_CALLSTATE -CONNECTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x2e(CallDeQueue) Caller = B Called = B HuntPilot = Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = B Called = HP1 HuntPilot = HP1 Connected = B

Table 41: Hunt List Call to HP1 and No Agents Logged In or Registered

Action	Expected events
App initiates call from A to HP1. (None of the Huntmembers are registered or logged in). Call is disconnected.	At A: LINE_CALLSTATE -DISCONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1

FailOver or FailBack Scenario

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on 1000:P1	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information

Action	Expected events
Stop Primary CTI Manager	OutofService for the line P1:1111 INService for the line P1:1111. **Note** There will not be any notification for the existing calls.

GroupCallPickup

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

Pickup group P1:1111 is configured and opened

P1:1000, P1:1001, P1:1002 are all in pickup group P1:1111

P1:2000, P1:2001, P1:2002 are all in pickup group P1:2222

P1:4000 and P1:4001 are configured

Action	Expected
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	LineGetDevCaps with Extension Version – 000A0000 on P1:2000CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information

Action	Expected
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupCallPickup option and GroupPickUp DN 1111 on P1:2000	Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupCallPickup option and GroupPickUp DN 1111 on P1:2000Events on P1:2000: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4000, Called = 1111, Connected = 4000, dwReason = Direct, dwOrigin = Internal **Note** There is no notification at P1:1111 after the call has been pickup.

OtherCallPickup

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

Pickup groups P1:1111, P1:2222, P1:3333 is configured and opened

P1:1000, P1:1001, P1:1002 are all in pickup group P1:1111

P1:2000, P1:2001, P1:2002 are all in pickup group P1:2222

P1:3000, P1:3001, P1:3002 are all in pickup group P1:3333

P1:1111, and P1:2222 are sub-groups, in order of priority, of pickup group P1:3333.

P1:4000 and P1:4001 are configured.

Action	Expected Event
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful

Action	Expected Event
P1:4000 calls P1:2000 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with OtherPickup option on P1:3000 **Note** Group DN is not required	Events on P1:3000: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4001, Called = 1000, Connected = 4001, dwReason = Direct, dwOrigin = Internal

DirectCallPickup

Action	Expected Event
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111

Action	Expected Event
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with DirectCallPickup option with pickup groupDN (1000) on P1:10001	Events on P1:1001: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 1001, Called = 1000, Connected = 4001, dwReason = Direct, dwOrigin = Internal

CallPickup (Negative Use Case)

Configuration

Service parameter Auto Call Pickup Enabled is enabled.

P1:2000 is already opened by the application.

Pickup groups P1:1111, P1:2222, P1:3333 is configured and opened.

P1:1000, P1:1001, P1:1002 are all in pickup group P1:1111

P1:2000, P1:2001, P1:2002 are all in pickup group P1:2222

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:2000	Line_Reply with Error LINEERR_OPERATIONUNAVAIL

GroupCallPickup with SuperSet Call PickupDN

Configuration

Service parameter Auto Call Pickup Enabled is enabled.

Pickup groups P1:1111, P1:2222, P1:3333 is configured and opened.

P1:1000, P1:1001, P1:1002 are all in pickup group P1:1111.

P1:2000, P1:2001, P1:2002 are all in pickup group P1:2222.

P1:3000, P1:3001, P1:3002 are all in pickup group P1:3333.

P1:1111, and P1:2222 are sub-groups, in order of priority, of pickup group P1:3333.

P1:4000 and P1:4001 are configured.

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:2000 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupPickup option with pickup group(3333) on P1:3000	Line_Reply with Error LINEERR_CALLUNAVAIL

Group or Direct CallPickup with Invalid DN

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful

Action	Expected events
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupPickup option with pickup group(9999) on P1:3000 Variant -Direct Call Pickup with InValid DN	Line_Reply with Error LINEERR_OPERATIONFAILED Line_Reply with Error LINEERR_INVALLINESTATE

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**.

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

Action	Events
A issues startrecording request by lineDevSpecific	Recording fails with error Response= CTIERR_SECURITY_CAPABLITY_MISMATCH as LINEERR_SECURITY_CAPABILITIES_MISMATCH

Scenario Two

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

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

Action	Events
A issues startrecording request by lineDevSpecific Recording session gets established between the agent phone and the recorder	Along with the regular events for call answer, the following events will also be delivered to the call observer: LINE_CALLDEVSPECIFIC hDevice=hCall-1 dwCallbackInstance=0 dwParam1= SLDSMT_RECORDING_ STARTED dwParam2=0 dwParam3=0

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

Action	Events
When B answers	Along with the regular events for call answer, the following events will also be delivered to the call observer: LINE_CALLDEVSPECIFIC hDevice=hCall-1 dwCallbackInstance=0 dwParam1= SLDSMT_RECORDING_ STARTED dwParam2=0 dwParam3=0

CCMEncryption Enhancements

Precondition: CTI service Parameter - "Require Public Key encryption" = true/false

Table 42: CiscoTSP Connecting to 10.x CUCM

Action	TAPI Messages	TAPI Structures
PhoneInitializeEx/LineInitializeEx	Devices are Enumerated/ Lines are Enumerated	

Note Applications would be able to control /monitor devices/Lines as before no change.

Variant: Test the same with Secure CUCM and Secure Connection between CiscoTSP and CTI.

Precondition: CTI service Parameter - "Require Public Key encryption" = False

Table 43: 9.x CiscoTSP Connecting to 10.x CUCM

Action	TAPI Messages	TAPI Structures	
PhoneInitializeEx/LineInitializeEx	Devices are Enumerated/ Lines are Enumerated		

Note Applications would be able to control /monitor devices/Lines as before no change

Precondition: CTI service Parameter - "Require Public Key encryption" = False

Table 44: 9.x CiscoTSP Connecting to 10.x CUCM

Action	TAPI Messages	TAPI Structures	
PhoneInitializeEx/LineInitializeEx	Initialization fails and CiscoTSP devices won't be Enumerated.	Notifier will pop-up error message indicating that Provider Init failed. Error - Provider Init failed - Incompatible protocol version	

CIUS Session Persistency

Notify the Line Application and Expose the Changed IP Address

Action	TAPI messages	TAPI structures
lineInitializeEx	lineDevices are Enumerated	
lineOpen for a lineDevice on the wireless device TAPI100	lineOpen() returns success	
lineGetDevCaps() with DeviceID = DeviceId of TAPI100	lineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.250" (FA1F4D0A -Little endian Hex format)

Action	TAPI messages	TAPI structures
The device TAPI100 moves across WiFi networks resulting in change in the IPv4 address from 10.77.31.250 to 10.77.31.176 **Variation 1:** The device TAPI100 moves from a IPv4 n/w to a Ipv6 n/w with new ip as 2001:db8::1:0:0:1 **Variation 2:** The device TAPI100 is docked/undocked and hence changes from WAN/LAN to wireless network	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_DEVICE_IPADDRESS Variation result: 1) Same as above 2) Same as above	
lineGetDevCaps() with DeviceID = DeviceId of TAPI100	lineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176" (B01F4D0A -Little endian Hex format) Variation 1: LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv6_only RegisteredIPv6Address = "2001:db8::1:0:0:1" (Application should use the Offset and size fields of IPv6 address from LINEDEVCAPS to retrieve the value of IPv6 address) Variation 2: LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176"

Notify the Phone Application and Expose the Changed IP Address

Action	TAPI Message	TAPI structures
phoneInitializeEx	phoneDevices are Enumerated	
phoneOpen for a phoneDevice of wireless device TAPI100	phoneOpen() returns success	

Action	TAPI Message	TAPI structures
phoneGetDevCaps() with DeviceID = DeviceId of TAPI100	phoneGetDevCaps() returns success	PHONEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.250" (FA1F4D0A -Little endian Hex format)
The device TAPI100 moves across WiFi networks resulting in change in the IPv4 address from 10.77.31.250 to 10.77.31.176 **Variation 1**: The deivce TAPI100 moves from a IPv4 n/w to a Ipv6 n/w with new ip as 2001:db8::1:0:0:1 **Variation 2**: The deivce TAPI100 is docked/undocked and hence changes from WAN/LAN to wireless network	EVENT = PHONE_DEVSPECIFIC dwParam1 = CPDSMT_PHONE_PROPERTY_ CHANGED_EVENT dwParam2 = PPCT_DEVICE_IPADDRESS Variation result: 1) Same as above 2) Same as above	
phoneGetDevCaps() with DeviceID = DeviceId of TAPI100	phoneGetDevCaps() returns success	PHONEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176" (B01F4D0A -Little endian Hex format) Phone Type = Cisco Cius. Phone Name = Cisco Phone [SEP123456789000] Variation 1: PHONEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv6_only RegisteredIPv6Address = "2001:db8::1:0:0:1" (Application should use the Offset and size fields of IPv6 address from PHONEDEVCAPS to retrieve the value of IPv6 address) Variation 2: PHONEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176" (B01F4D0A -Little endian Hex format)

Click to Conference

Third-party conference gets created by using click-2-conference feature:

Action	Events
Use Click-to-Call to create call from A to B, and B answers	For A: CONNECTED reason = DIRECT Calling = A, Called = B, Connected = B For B: CONNECTED reason = DIRECT Calling = A, Called = B, Connected = A

Action	Events
Use Click-2-Conference feature to add C into conference, and C answers	For A: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = B CONFERENCED Calling = A, Called = C, Connected = C For B: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = A CONFERENCED Calling = B, Called = C, Connected = C For C CONNECTED Reason = UNKNOWN ExtendedCallReason = ClickToConference CONFERENCED Calling = C, Called = A, Connected = A CONFERENCED Calling = C, Called = B, Connected = B

Creating Four-Party Conference by Using Click-2-Conference Feature

Action	Events
Use Click-to-Call to create call from A to B	For A: CONNECTED reason = DIRECT Calling = A, Called = B, Connected = B For B: CONNECTED reason = DIRECT Calling = A, Called = B, Connected = A
Use Click-2-Conference feature to add C into conference	For A: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = B CONFERENCED Calling = A, Called = C, Connected = C For B: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = A CONFERENCED Calling = C, Called = C, Connected = C For C CONNECTED Reason = DIRECT ExtendedCallReason = ClickToConference CONFERENCED Calling = C, Called = A, Connected = A CONFERENCED Calling = C, Called = B, Connected = B

Action	Events
Use Click-2-Conference feature to add party D	

Action	Events
	For A: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = B CONFERENCED Calling = A, Called = C, Connected = C CONFERENCED Calling = A, Called = D, Connected = D For B: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = A CONFERENCED Calling = B, Called = C, Connected = C CONFERENCED Calling = B, Called = D, Connected = D For C CONNECTED Reason = UNKNOWN ExtendedCallReason = ClickToConference CONFERENCED Calling = C, Called = A, Connected = A CONFERENCED Calling = C, Called = B, Connected = B CONFERENCED Calling = C, Called = D, Connected = D For D CONNECTED Reason = UNKNOWN

Action	Events
	ExtendedCallReason = ClickToConference
	CONFERENCED Calling = D, Called = A, Connected = A CONFERENCED Calling = D, Called = B, Connected = B CONFERENCED Calling = D, Called = C, Connected = C

Drop Party by Using Click-2-Conference

Action	Events
Conference gets created by using Click-2-Conference feature to add C into conference	For A: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = B CONFERENCED Calling = A, Called = C, Connected = C For B: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = A CONFERENCED Calling = B, Called = C, Connected = C For C CONNECTED Reason = UNKNOWN ExtendedCallReason = ClickToConference CONFERENCED Calling = C, Called = A, Connected = A CONFERENCED Calling = C, Called = B, Connected = B

Action	Events
Drop C from Click-2-Conference feature	For A CONNECTED Reason = DIRECT ExtendedCallReason = DIRECT Calling = A, Called = B, Connected = B For B CONNECTED Reason = DIRECT ExtendedCallReason = DIRECT Calling = A, Called = B, Connected = A For C IDLE

Drop Entire Conference by Using Click-2-Conference Feature

Action	Events
Conference gets created by using Click-2-Conference feature to add C into conference	For A: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = B CONFERENCED Calling = A, Called = C, Connected = C For B: CONNECTED reason = DIRECT ExtendedCallReason = DIRECT CONFERENCED Calling = A, Called = B, Connected = A CONFERENCED Calling = B, Called = C, Connected = C For C CONNECTED Reason = UNKOWN ExtendedCallReason = ClickToConference CONFERENCED Calling = C, Called = A, Connected = A CONFERENCED Calling = C, Called = B, Connected = B
Drop entire conference	For A IDLE For B IDLE For C IDLE

Conference Enhancements

Noncontroller Adding Parties to Conferences

A,B, and C exist in a conference that A created.

Action	Events
A,B, and C exist in a conference	At A: Conference – Caller = A, Called = B, Connected = B Connected Conference – Caller = A, Called = C, Connected = C At B: Conference – Caller = A, Called = B, Connected = A Connected Conference – Caller = B, Called = C, Connected = C At C: Conference – Caller = B, Called = C, Connected = B Connected Conference – Caller = C, Called = A, Connected = A
C issues a linePrepareAddToConference to D	At A: Conference – Caller = A, Called = B, Connected = B Connected Conference – Caller = A, Called = C, Connected = C At B: Conference – Caller = A, Called = B, Connected = A Connected Conference – Caller = B, Called = C, Connected = C At C: Conference – Caller = B, Called = C, Connected = B OnHoldPendConf Conference – Caller = C, Called = A, Connected = A Connected -Caller = C, Called = D, Connected = D At D: Connected -Caller = C, Called = D, Connected = C

Action	Events
C issues a lineAddToConference to D	At A: Conference – Caller = A, Called = B, Connected = B Connected Conference – Caller = A, Called = C, Connected = C Conference – Caller = A, Called = D, Connected = D At B: Conference – Caller = A, Called = B, Connected = A Connected Conference – Caller = B, Called = C, Connected = C Conference – Caller = B, Called = D, Connected = D At C: Conference – Caller = B, Called = C, Connected = B Connected Conference – Caller = C, Called = A, Connected = A Conference – Caller = C, Called = D, Connected = D At D: Conference – Caller = C, Called = D, Connected = C Connected Conference – Caller = D, Called = A, Connected = A Conference – Caller = D, Called = B, Connected = B

Chaining Two Ad Hoc Conferences Using Join

Actions	TSP CallInfo
A calls B, B answers, then B initiates conference to C, C answers, and B completes the conference	At A: GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = A Called = B CONFERENCED : Caller = A Called = C At B: GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = A Called = B CONFERENCED : Caller = B Called = C At C: GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = B Called = C CONFERENCED : Caller = C Called = A

Actions	TSP CallInfo
C initiates or completes conference to D and E	

Actions	TSP CallInfo
	No Change for A and B At C: -First conference GCID-1 ONHOLD : Caller = Unknown Caller = Unknown CONFERENCED : Caller = A Called = B CONFERENCED : Caller = A Called = C -Second conference GCID-2 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = C Called = D CONFERENCED : Caller = C Called = E At D: GCID-2 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = C Called = D CONFERENCED : Caller = D Called = E At E: GCID-2 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = C Called = E CONFERENCED : Caller = E

Actions	TSP CallInfo
	Called = D
C initiates JOIN request to join to conference call together, with GCID as the primary call	At A: GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = A Called = B CONFERENCED : Caller = A Called = C CONFERENCED : Caller = A Called = Conference-2 At B : GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = A Called = B CONFERENCED : Caller = B Called = C CONFERENCED : Caller = B Called = Conference-2 At C: -First conference GCID-1 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = B Called = C CONFERENCED : Caller = C Called = A CONFERENCED : Caller = C Called = Conference-2

Actions	TSP CallInfo
	At D: GCID-2 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = D Called = E CONFERENCED : Caller = D Called = Conference-1 At E : GCID-2 CONNECTED : Caller = Unknown Caller = Unknown CONFERENCED : Caller = E Called = D CONFERENCED : Caller = E Called = Conference-1

CTI Remote Device

Expose Remote Destination Info for CTI Remote Device in ProviderDeviceLineInfoEvent

PreCondition: User has a CTI remote device "CTIRD1" under it control list. CTIRD1 device has 3 remote destinations configured.

Action	CTI messages/Events
Application opens the provider.	CTI acquires the devices which are under control list of the user
Application sends GetSignleDeviceAndLineInfoRequest to CTI to fetch info for CTIRD1 device.	CTI sends ProviderDeviceLineInfoEvent to application and exposes 3 RDs configured on the device as part of "Remote Destination Info" structure.

Expose Remote Destination Info for CTI Remote Device in ProviderDeviceRegisteredWithLineInfoNotify

PreCondition: User has a CTI remote device "CTIRD1" under it control list. CTIRD1 device has 3 remote destinations configured.

Action	CTI messages/Events
Application opens the provider.	CTI acquires the devices which are under control list of the user
Application sends GetSignleDeviceAndLineInfoRequest to application to fetch info for CTIRD1 device.	CTI sends ProviderDeviceLineInfoEvent to application and exposes 3 RDs configured on the device as part of "Remote Destination Info" structure.
Application resets the device CTIRD1 from the admin page.	CTI sends ProviderDeviceRegisteredWithLineInfoNotify to application and exposes 3 RDs configured on the device as part of "Remote Destination Info" structure.

Expose New Device Type for CTI Remote Device

Precondition:

CTIRD (CTI Remote Device -Name: CTIRDdrajesh)

Remote Destinations configured/will be configured on CTI Remote Device:

RD1-CTIRD -(Name: Mobile, Number: 914086271309)

RD2-CTIRD -(Name: Office, Number: 914089022131)

Line-A (DN -1000) -Line-A configured on CTI Remote Device (shared line of Enterprise DN -1000 configured on Device EP)

EP (Enter Prise Phone -SCCP -IP Phone)

Line-A' -DN -1000 configured on Device EP

CSF (CSF Device -Name: CSFdrajesh)

Line-A" -DN -1000 configured on Device CSF

Remote Destination configured on CSF device:

RD1-CSF -(Name: CSF-Mobile, Number: 914086271310)

RD2-CSF -(Name: CSF-Office, Number: 914089022132)

Action	TAPI messages	TAPI structures
PhoneInitializeEx	Devices are Enumerated	
PhoneGetDevCaps() with DeviceID = DeviceId of CTIRD.	PhoneGetDevCaps() returns success	PHONECAPS::PhoneInfo = "CTI Remote Device" PHONECAPS:: PhoneName = "Cisco Phone: [CTIRDdrajesh]"
PhoneGetDevCaps() with DeviceID = DeviceId of CSF.	PhoneGetDevCaps() returns success	PHONECAPS::PhoneInfo = "Cisco Unified Client Services Framework" PHONECAPS:: PhoneName = "Cisco Phone: [CSF-drajesh]"

Enumerating CTI Remote Devices and Exposing Remote Destination Information to Application

Precondition: same as above usecase; RD1-CTIRD and RD1-CSF are configured on respective devices

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A" on CSF.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "91486271310" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Add Remote Destination From Admin and Expose Multiple Remote Destination Information to Application

Precondition: In addition to above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A on CTIRD	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Add other Remote Destination RD2-CTIRD on CTI Remote Device from Admin Pages RD2-CTIRD Info -(Name: Office, Number: 4089022131)	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "91486271310" isActiveRD = 0x00000000 unicodeRDName = "CSF-Office" RDNumber = "4089022132" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Update RD Info (RDName/Number/Both) From Admin -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A on CTIRD	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Update Remote Destination RD2-CTIRD Name on CTI Remote Device "CTIRD" from Admin Pages RD2-CTIRD Info -(Name: Home, Number: 4089022132)	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Home" RDNumber = "4089022132" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Update Remote Destination RD2-CTIRD Number on CTI Remote Device CTIRD from Admin Pages RD2Info -(Name: Home, Number: 4089021234)	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID =	LineGetDevCaps() returns	LINEDEVCAPS::DevSpecific
LineDeviceId of Line-A on CTIRD.	success	Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Home" RDNumber = "4089021234" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
Update Remote Destination RD2-CTIRD Name and Number on CTI Remote Device CTIRD from Admin Pages RD2Info -(Name: Office, Number: 4089022131)	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000) EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info with respective RD Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Remove RD From Admin -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Remove Remote Destination RD2-CTIRD on CTI Remote Device CTIRD from Admin Pages RD2Info -(Name: Office, Number: 4089022131)	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Remote Destination Information on CTI RemoteDevice/CSF Device Which Does Not Have Remote Destination's Configured

Precondition: In addition to above usecase

CTIRD2 (CTI remote device -doesn't have any RemoteDestination's configured)

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-C on CTIRD2.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info is empty RemoteDestinationOffset = 0 RemoteDestinationSize = 0 RemoteDestinationCount = 0 RemoteDestinationElementFixedSize = 0 IsMyAppLastToSetActiveRD = 0x00000000

Remote Destination Information on Non CTI RemoteDevice / CSF Device

Precondition: In addition to above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A' on EP.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific DeviceProtocolType = DeviceProtocolType_SCCP (0x01) Remote Destination Info is empty RemoteDestinationOffset = 0 RemoteDestinationSize = 0 RemoteDestinationCount = 0 RemoteDestinationElementFixedSize = 0 IsMyAppLastToSetActiveRD = 0x00000000

Add RD From Application -RD Info Change Notification to Application

Precondition: Remove All RD's from Admin Page

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: RemoteDestinationOffset = 0 RemoteDestinationSize = 0 RemoteDestinationCount = 0 RemoteDestinationElementFixedSize = 0 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A Add Remote Destination RD2-CTIRD to CTI Remote Device CTIRD: CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "4089022131" m_UnicodeRDName = "Office" m_activeRD = 0x00000000	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE LINE_REPLY with success EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Update RD Info (RDNumber/RDName/Both) From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Update Remote Destination name of RD2-CTIRD on CTI Remote Device "CTIRD": CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "4089022131" m_UnicodeRDName = "Office-Change" m_NewRDNumber = "4089022131" m_activeRD = 0x00000000	LINE_REPLY with success EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office-Change" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
Update Remote Destination Number of RD2-CTIRD on CTI Remote Device "CTIRD": CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "4089022131" m_UnicodeRDName = "Office-Change" m_NewRDNumber = "4089020000" m_activeRD = 0x00000000	LINE_REPLY with success EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office-Change" RDNumber = "4089020000" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Update Remote Destination Name and Number of RD2-CTIRD on CTI Remote Device "CTIRD": CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "408902000" m_UnicodeRDName = "Office" m_NewRDNumber = "4089022131" m_activeRD = 0x00000000	LINE_REPLY with success EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000) EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Update RD Info (SetActive RD) From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_REMOTE_ DEVICE(0x03) Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
Set RD2-CTIRD as ActiveRD: Req CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "4089022131" m_UnicodeRDName = "Office" m_RDNumber = "4089022131" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001
LineShutdown()	LineShutdown success	
Active RD will be RESET to False when the Application which has set RD as ACTIVE is shutdown or closed		
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Add Other RD (RD2-CTIRD with IsActive Set) From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A on CTIRD	LineOpen() returns Success	
Set RD2-CTIRD -"Office" as ACTIVE		
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001
Add Remote Destination RD1-CTIRD on CTI Remote Device CTIRD with "IsActive" set to true CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001

Action	TAPI messages	TAPI structures
Variation : Add RD1-CTIRD with IsActive RD = False	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Update RD (RD1-CTIRD -Name, Number and Set IsActive) From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase Variation (RD2 is added with IsActive = false)

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Set RD2-CTIRD-"Office" as ACTIVE		

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001
Update Remote Destination RD1-CTIRD on CTI Remote Device "CTIRD" with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile-t" m_NewRDNumber = "91408627130900" m_activeRD = 0x00000001	*** 2 Change Nofitications EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000) EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile-t" RDNumber = "9148627130900" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001

Action	TAPI messages	TAPI structures
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Remove RD (RD1-CTIRD Which Is Active RD) From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Set RD1-CTIRD-"Mobile-t" as ACTIVE		
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile-t" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
Remove Remote Destination RD1-CTIRD on CTI Remote Device "CTIRD" CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "9148627130900"	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Negative -Add RD From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A of CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
Add Remote Destination on CTI Remote Device CTIRD Variation 1: Empty RD Number : m_RDNumber = "" CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "" m_UnicodeRDName = "" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_INVALPARAM	
Variation 2: RDNumber : same RD Number as any of the existing RD's Name "12345" -RD already configured on CUCM. CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "12345" m_UnicodeRDName = "Office" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_DUPLICATE_INFORMATION (0xC0000013)	
Variation 3: Add RD when the user Limit for UserID used for CTI RD is reached. For example : if User has limit set to 4 and then if Remote Device is already configured with 4 Remote Destination and User tries to Add 5th one from Application. CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "12345" m_UnicodeRDName = "temp" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_REMOTE_DESTINATION_LIMIT_EXCEEDED (0xC0000015)	

Action	TAPI messages	TAPI structures
Variation 4: RDNumber : Invalid Remote Destination Name [name has unsupported characters, eg-name&] or invalid number [cant configure any of the local device DN as number which is not supported] CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "1000" m_UnicodeRDName = "Office&" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_INVALPARAM	
Variation 5: Add RD to a CSF device which doesn't have Owner/END User ID configured CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "12345" m_UnicodeRDName = "Office" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE (0xC000001B)	
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Negative -Update RD From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Update Remote Destination on CTI Remote Device: Variation 1: Empty RD Number : m_RDNumber = "" CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "" m_UnicodeRDName = "" m_NewRDNumber = "" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_INVALPARAM	
Variation 2: RDNNumber : RD Number in Request doesn't match with any of the existing RD in the RD List on Device CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "12345" m_UnicodeRDName = "Temp" m_RDNumber = "12345" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_REMOTE_DESTINATION_UNAVAIL (0xC0000014)	

Action	TAPI messages	TAPI structures
Variation 3: RDNaumber : same RD Number as any of the existing RD's Name *** RDNumber "4086271309" is already configured on other RemoteDestination CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "4089022131" m_UnicodeRDName = "Office" m_RDNumber = "4086271309" m_activeRD = 0x00000000	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR _DUPLICATE_INFORMATION (0xC0000013)	
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Negative -Remove RD From Application -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A on CTIRD.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Remove Remote Destination on CTI Remote Device: Empty RDNumber : CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = ""	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_INVALPARAM	
Variation 1: RDNumber : RD Number in Request doesn't match with any of the existing RD in the List CiscoLineDevSpecific AddRemoteDestination Req m_RDNumber = "1234567"	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_REMOTE_DESTINATION_UNAVAIL (0xC0000014)	
Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Negative -Add/remove/update RD From Application -on Non-CTI RD /CSF Device Line or Line Is Not Opened with Required Extension

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Add/Remove/Update Remote Destination on CTI Remote Device CTIRD Variation 1: Previous step Line is not opened with required ext Version -(0x000C0000 or greater)	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_OPERATIONUNAVAIL	
Variation 2: Req on Line which is not on CTI Remote Device / CSF device	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_OPERATIONUNAVAIL	
Variation 3: Failure of Add/Remove/update Req for any other reasons not captured in above useCases	LineDevSpecific() returns dwRequestID LINE_REPLY lResult = LINEERR_OPERATIONFAILED	

Multiple Apps Setting Active RD

Precondition: same as UseCase 1

Action	TAPI messages	TAPI structures
App1 and App2: LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
App1 and App2: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
App1 and App2: LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
App1: Update Remote Destination RD2 on CTI Remote Device "CTIRD" with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	Change Notification to App1 and App2: EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
App1: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
App2: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
App2: Update Remote Destination RD2 on CTI Remote Device "CTIRD" with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914089022131" m_UnicodeRDName = "Office" m_NewRDNumber = "914089022131" m_activeRD = 0x00000001	Change Notification to App1 and App2: EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
App1: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000000

Action	TAPI messages	TAPI structures
App2: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001
Variant 1: App2: LineShutdown()	LineShutdown() returns success Change Notification to App1: EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000	

Action	TAPI messages	TAPI structures
App1: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Variant 2: App1: LineShutdown()	LineShutdown() returns success No Change Notification to App2	
App2: LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000001 IsMyAppLastToSetActiveRD = 0x00000001

Action	TAPI messages	TAPI structures
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

CTI/CCM Manager FailOver Scenario - Active RD

Precondition: same as UseCase 1

TSP is configured with Primary and Secondary CTI Manager

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
Update Remote Destination RD1 on CTI Remote Device "CTIRD" with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
Stop Primary CTI Manager TSP connects to Secondary CTIManager and Active RD configuration is RE-SET by CiscoTSP	Event on Line A : Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
Set RD -Mobile to ACTIVE RD and then Stop Call Manager on the node of Secondary CTI Manager ActiveRD configuration is not changed/ not RESET	Event on Line A : Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

CTI/CCM Manager FailOver Scenario - Active RD Set by Other Application

Precondition: same as UseCase 1

TSP is configured with Primary and Secondary CTI Manager

Other Application has set the ACTIVE RD on the Device and Application is connected to Secondary CTI Manager

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A	LineOpen() returns Success Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Stop Primary CTI Manager Active RD configuration is not RESET as the this Application has not set the ACTIVE RD	Event on Line A : Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = LINEDEVCAPSDEVSPECIFIC_ REMOTEDEVICE (0x00000008) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE(0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "91486271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Stop Call Manager on the node of Secondary CTI Manager ActiveRD configuration is not changed/ not RESET	Event on Line A : Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
Variation : Test the same on CSF device [CSF -Line-A"]		Same as for CTI Remote Device other than dwLineTypes and DeviceProtocolType Info. dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02)

Monitoring CSF Device in Soft Phone/Desk Phone Mode

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A" on CSF Device.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A"	LineOpen() returns Success	
LineSetStatusMessages()on Line-A" with dwLineStates = INSERVICE and OUTOFSERVICE	Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
LineMake Call() or any Incoming Call	Call Events are reported to Application	
Lineclose and ShutDown	LineClose and LineShutdown Success	

Monitoring CSF Device Switching Mode From Soft/Desk Phone Mode to Extend Mode

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A" on CSF device.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A"	LineOpen() returns Success	
LineSetStatusMessages() on Line-A" with dwLineStates = INSERVICE and OUTOFSERVICE	Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	
From Jabber Client Switch the mode to Extend Mode	Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_DEVICE_PROTOCOL_TYPE (0x00008000)	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A".	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000
Lineclose and ShutDown	LineClose and LineShutdown Success	

Monitoring CSF Device in Extend Mode, Switches Back to Soft / Desk Phone Mode

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A" on CSF device.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
LineOpen() with ExtVer-0x000C0000 dwDeviceID = LineDeviceID of Line-A"	LineOpen() returns Success	
LineSetStatusMessages()on Line-A" with dwLineStates = INSERVICE and OUTOFSERVICE	Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE	

Action	TAPI messages	TAPI structures
From Jabber Client Switch the mode to Soft Mode Or From Jabber Client Switch the mode to Deskphone Mode	Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_OUTOFSERVICE Line INSERVICE EVENT Event = LINE_LINEDEVSTATE dwParam1 = LINEDEVSTATE_INSERVICE EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_DEVICE_PROTOCOL_TYPE (0x00008000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A".	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_SIP (0x02) Remote Destination Info: unicodeRDName = "CSF-Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Lineclose and ShutDown	LineClose and LineShutdown Success	

Basic Incoming Call to CTI Remote Device

CTI remote device:

A (CTI Remote Device -Name: CTIRD1)

Remote Destination:

RD1 -Remote Destination configured on CTI Remote Device A

(Name: Mobile, Number: 914086271309)

RD2 -Remote Destination configured on CTI Remote Device A

(Name: Office, Number: 914089022131)

Line:

Line-A1 (DN -2000) (Alerting Name:2000name, Display Name: CTIRD-2000name) configured on CTI Remote Device A (shared line of Enterprise DN -2000 configured on Device B)

Line-A2 (DN -2001) (Alerting Name:2001name, Display Name: CTIRD-2001name) configured on CTI Remote Device A (shared line of Enterprise DN -2001 configured on Device B)

Enterprise Phones:

B (IP Phone -Name: SEPxxxxxxxx)

Line:

Line-A1' -DN -2000(Alerting Name: 2000name, Display Name: EP-2000name) configured on Device B

Line-A2' -DN -2001(Alerting Name: 2001name, Display Name: EP-2001name) configured on Device B

C (IP Phone -Name: SEPxxxxxxxx)

Line:

Line-C -DN -1000(Alerting Name: 1000name, Display Name: 1000Name) configured on Device C

D (IP Phone -Name: SEPxxxxxxxx)

Line:

Line-D -DN -1001(Alerting Name: 1001name, Display Name: 1001Name) configured on Device D

CSF Device:

D (CSF Device -Name: CSF-drajesh)

Remote Destination:

RD-01 -Remote Destination configured on CSF device D

(Name: CSF-Mobile, Number: 914086271309)

RD-02 -Remote Destination configured on CSF device D

(Name: CSF-Office, Number: 914089022131)

Line:

Line-A" (DN -2000) -Line-A (Alerting Name: 2000name, Display Name: CSF-2000) configured on CSF device D (shared line of Enterprise DN -2000 configured on Device B)

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Open all Lines (A, A' and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
LineMakeCall on Line-C with DN (A -DN 2000)	LineMakeCall() success Call on C : LINE_CALLSTATE -Param1 = DIALING LINE_CALLSTATE -Param1 = PROCEEDING LINE_CALLSTATE -Param1 = RINGBACK Call on CTI Remote Device : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Call on Enterprise Phone : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED	

Action	TAPI messages	TAPI structures
After "Delay Before Ringing Timer" expires the call is offered on Remote Destinations and all Remote Destinations Ring		
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID =
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID =

Action	TAPI messages	TAPI structures
Answer on any of the Remote Destination	Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED (active) Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = CTIRD-2000name ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID = 2000

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = CTIRD-2000name ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID = 2000
LineDrop() for the call on Device A (CTI-RD) *** Call on Remote Destination is dropped	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	

Action	TAPI messages	TAPI structures
Variation : Answer the call on Enterprise Phone (B) LineAnswer() on the call on Device B *** Call on Remote Device/Remote Destination drops	Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED	
Variation : One of the Remote Destination answers the call before the "Answer Too Soon Timer"	Expected Result : All calls go to Disconnected/IDLE State	
Variation : Active RD set on CTI Remote Device	Expected result: only Remote Destination which is set ACTIVE rings Call rings immediately and "Delay before Ringing Timer" wouldn't be effective when ACTIVE RD is set. Remote Destination can answer the call Immediately and "Answer Too Soon Timer" wouldn't be effective when ACTIVE RD is set.	
Continuation to above variation On second Incoming Call...	There won't be second call on Remote Destination, only at Remote Device second call will present and reported to Application.	
Variation : Test with CSF Device in Extend Mode	Expected result: would be same as observed on CTI Remote Device	

DVO Call (Outgoing Call Initiation From CTI Remote Device)

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Open all Lines (A, A' and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
LineMakeCall on Line-A with DN (C -DN 1000)	LineMakeCall() returns RequestID LINE_REPLY Param1 = RequestID Param2 = LINEERR_OPERATION_FAIL_NO_ACTIVE_RD_SET (0xC0000016)	
Update Remote Destination RD1 "Mobile"on CTI Remote Device A with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "4086271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
LineMakeCall on Line-A with DN (C -DN 1000) *** Only Remote Destination "Mobile" rings and it rings immediately as the RD is set Active *** No Call presented on EP	LineMakeCall() success Call on CTI Remote Device : LINE_CALLSTATE -Param1 = OFFERING	
Answer the first Call on CTI Remote Device: Answer() on the call on CTIRemote Device(A)	LineAnswer() fail with Error LINEEE_OPERATIONUNAVAIL	

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A(CTIRD)	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 2000 dwCallerIDName = voiceConnect dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = ModifiedCallingParty = 2000 ModifiedCalledParty = 2000 ModifiedConnectedID =
Once Remote Destination answers the call, call will be offered on initial dialed number C Call will be present on Enterprise Phone and call will be Remote In Use Call	Call on C : LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED LINE_CALLSTATE -Param1 = RINGBACK Call on Enterprise Phone : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = ACCEPTED LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	

Action	TAPI messages	TAPI structures
C answers the call LineAnswer() on call on Device-C	LineAnswer() success Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED (active) Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: CallReason = UNKNOWN (0x400) dwCallerID = 2000 dwCallerIDName = 2000name dwCalledID = 1000 dwCalledIDName = 1000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: ExtendedCallReason = CtiReasonMobility(0x021 = 33) UnicodeCallerPartyName = 2000name UnicodeCalledPartyName = 1000name UnicodeConnectedPartyName = 2000name ModifiedCallingParty = 2000 ModifiedCalledParty = 1000 ModifiedConnectedID = 2000

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 2000 dwCallerIDName = 2000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 1000 dwConnectedIDName = 1000name DevSpecific :: CallAttributeType = TSPCallAttribute_DVOCall (0x00002000) UnicodeCallerPartyName = 2000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = 1000name ModifiedCallingParty = 2000 ModifiedCalledParty = 2000 ModifiedConnectedID = 1000
LineDrop() for the call on Device A (CTI-RD)	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Multiple Calls -Answer/Hold/Resume

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "4086271309" isActiveRD = 0x00000000 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000000
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Update Remote Destination RD1 "Mobile"on CTI Remote Device A with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
Make Call between C and A[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same as above test cases		

Action	TAPI messages	TAPI structures
LineMakeCall on Line-D with DN (A -DN 2000)	LineMakeCall() success Call on Device-D : LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Second Call on CTI Remote Device[A] [D ' A] : LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Second Call on Enterprise Phone[B] [D ' A]: LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED	
There won't be second call offered to Remote Destination		
Answer() on the second call on CTIRemote Device(A) Remote Destination and D will be talking/ will have Media connection	LineAnswer() returns success Calls on CTI Remote Device : Call1 [C ' A]: LINE_CALLSTATE -Param1 = ONHOLD Call1 [D ' A]: LINE_CALLSTATE -Param1 = CONNECTED Calls on Enterprise Phone[B] : Call1 [C ' A]: LINE_CALLSTATE -Param1 = ONHOLD Call1 [D ' A]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	

Action	TAPI messages	TAPI structures
Resume the first call on CTIRemote Device [A] LineUnhold() on the call [c ' A] on Device A Remote Destination and C will be talking/ will have Media connection	LineUnHold() returns success Calls on CTI Remote Device : Call1 [C ' A]: LINE_CALLSTATE -Param1 = CONNECTED Call1 [D ' A]: LINE_CALLSTATE -Param1 = ONHOLD Calls on Enterprise Phone[B] : Call1 [C ' A]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Call1 [D ' A]: LINE_CALLSTATE - Param1 = ONHOLD	
Resume the ONHOLD call [D ' A]from Enterprise Phone LineUnHold() on the call [D ' A] on Device B	LineUnHold() returns success Calls on CTI Remote Device : Call1 [C ' A]: LINE_CALLSTATE -Param1 = CONNECTED Call1 [D ' A]: LINE_CALLSTATE -Param1 = IDLE Calls on Enterprise Phone[B] : Call1 [C ' A]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Call1 [D ' A]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x01(active)	

Action	TAPI messages	TAPI structures
LineDrop() for the call on Device A (CTI-RD) Call on Remote Destination will be dropped	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Multiple Calls -Multiple Lines -Answer/Hold/Resume

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Update Remote Destination RD1 "Mobile"on CTI Remote Device A with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	

Action	TAPI messages	TAPI structures
Make Call between C and A[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		
LineMakeCall on Line-D with DN (A2 -DN 2001)	LineMakeCall() success Call on Device-D : LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Second Call on CTI Remote Device[A] [D ' A2]: LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Second Call on Enterprise Phone[B] [D ' A2]: LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED	
There won't be second call offered to Remote Destination		
Answer() on the second call on CTIRemote Device(A) Remote Destination and D will be talking/ will have Media connection	LineAnswer() returns success Calls on CTI Remote Device : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = ONHOLD Call1 [D ' A2]: LINE_CALLSTATE -Param1 = CONNECTED Calls on Enterprise Phone[B] : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = ONHOLD Call1 [D ' A2]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	

Action	TAPI messages	TAPI structures
Resume the first call on CTIRemote Device [A] LineUnhold() on the call [c ' A1] on Device A Remote Destination and C will be talking/ will have Media connection	LineUnHold() returns success Calls on CTI Remote Device : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = CONNECTED Call1 [D ' A2]: LINE_CALLSTATE -Param1 = ONHOLD Calls on Enterprise Phone[B] : Call1 [C ' A1]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Call1 [D ' A2]: LINE_CALLSTATE - Param1 = ONHOLD	
Drop the Connected Active Call on CTI Remote Device. LineDrop() for the call[C ' A1] on Device A (CTI-RD) Call on Remote Destination will not be dropped as there is other Active/OnHold call on CTI Remote Device As second Call is on OnHold state, Remote Destination will listen Dead Air	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Calls on CTI Remote Device : [C ' A1] : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : Call [C ' A1] LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	

Action	TAPI messages	TAPI structures
Drop the onHold call on CTI Remote Device LineDrop() for the call on Device A (CTI-RD) Call on Remote Destination is dropped C and EP call will not be disconnected. On C call will be in Connected state and on EP call will be in OnHold state.	LineDrop() success Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Transfer

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same as above test cases		
Setup Transfer and Dial D LineSetupTransfer() on the call [C ' A1] on Device A	LineSetupTransfer returns success Primary Call on CTI Remote Device[A] [C ' A1] : LINE_CALLSTATE -Param1 = OnholdPendingTransfer Consult Call on CTI Remote Device[A] [A1 ' D]:	

Action	TAPI messages	TAPI structures
LineDial() on Consult call with DN -D	LINE_CALLSTATE -Param1 = DIALTONE LINE_CALLSTATE -Param1 = DIALING Calls on Enterprise Phone[B] : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = ONHOLD Call1 [A1 ' D]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Consult Call on CTI Remote Device[A] [A1 ' D]: LINE_CALLSTATE -Param1 = PROCEEDING LINE_CALLSTATE -Param1 = RINGBACK	
Answer the Call on Device D Remote Destination and D will be talking/ will have Media connection	Secondary Call on CTI Remote Device: Call1 [A1 ' D]: LINE_CALLSTATE -Param1 = CONNECTED Param2 = 0x01(active)	
Complete Transfer on the Primary Call[C ' A]with [A ' D] call as consult call LineCompleteTranfer() on the call [c ' A1] on Device A D and C will be talking/ will have Media connection	Both the Calls on CTI Remote Device Drop Primary Call on CTI Remote Device : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Secondary Call on CTI Remote Device: Call1 [A ' D]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Direct Transfer on Same Line

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		
Make Call between D and A1 Call Info is same above Multiple Call across lines test case		
DirectTrnasfer on the calls on CTI Remote Device Both Calls on Remote Device and call on Remote Destination drop	Both the Calls on CTI Remote Device Drop Primary Call on CTI Remote Device : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Secondary Call on CTI Remote Device: Call1 [A1 ' D]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE	
DirectTrnasfer on the calls on CTI Remote Device Both Calls on Remote Device and call on Remote Destination drop CciscoLineDevSpecificDirectTransfer on the call [c ' A1] on Device A with ConsultCallID = CallID of [D ' A1] D and C will be talking/ will have Media connection	Both the Calls on CTI Remote Device Drop Primary Call on CTI Remote Device : Call1 [C ' A1]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Secondary Call on CTI Remote Device: Call1 [A1 ' D]: LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE	

Action	TAPI messages	TAPI structures
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Conference -Setupconference/AddtoConference

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		

Action	TAPI messages	TAPI structures
Setup Conference and Dial D LineSetupConference() on the call [C ' A1] on Device A LineDial() on Consult call with DN -D	LineSetupConference returns success Original Call on CTI Remote Device[A] : LINE_CALSTATE = CONFERENCE Conference Parent Call on CTI Remote Device[A] : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = OnholdPendingConference Consult Call on CTI Remote Device[A] : LINE_CALLSTATE -Param1 = DIALTONE LINE_CALLSTATE -Param1 = DIALING Calls on Enterprise Phone[B] : Call1 [C ' A]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Call1 [A ' D]: LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) Consult Call on CTI Remote Device[A] : LINE_CALLSTATE -Param1 = PROCEEDING LINE_CALLSTATE -Param1 = RINGBACK	
Answer the Call on Device D Remote Destination and D will be talking/ will have Media connection	Secondary Call on CTI Remote Device: Call1 [A ' D]: LINE_CALLSTATE -Param1 = CONNECTED LINE_CALLSTATE -Param1 = IDLE	

Action	TAPI messages	TAPI structures
Complete Conference on the Primary Call[C ' A]with [A ' D] call as consult call LineAddtoConference() on the call [c ' A1] on Device A All 3 parties C, D and CTI Remote Device[Remote Destination] will be in Conference	Call model on CTI Remote Device : [C ' A1]-[Original Call1]-[state = Conference] [A1 ' Conference]-[Conference Parent Call]-[State = CONNECTED] [A1 ' D]-[Consult Call]-[state -CONFERENCE] Call Model on Enterprise Phone: Same as CTI Remote Device, all calls are RIU Calls	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Join on Same Line

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		
Make Call between D and A1 Call Info is same above Multiple Call across lines test case		

Action	TAPI messages	TAPI structures
Join on the Primary Call[C ' A1]with [A1 ' D] call as consult call CCiscoLineDevSpecificJoin() on the call [c ' A1] on Device A with CallIDstoJoin = CallID of Call [D ' A1] CTIRemoteDevice [A -Remote Destination], D and C will be in Conference.	Original Call on CTI Remote Device[A] [C ' A1]: LINE_CALSTATE = CONFERENCE Conference Parent Call on CTI Remote Device[A] : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = CONNECTED Consult Call on CTI Remote Device[A] [D ' A1]: LINE_CALLSTATE -Param1 = CONFERENCE Conference Model will be created on CTI Remote Device and RIU Conference Model on EP	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Direct Transfer/Join Across Line on CTI Remote Device

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		
Make Call between D and A2 Call Info is same above Multiple Call across lines test case		

Action	TAPI messages	TAPI structures
Join on the Primary Call[C ' A1]with [A2 ' D] call as consult call CCiscoLineDevSpecificJoin() on the call [c ' A1] on Device A with CallIDstoJoin = CallID of Call [D ' A2] Or CciscoLineDevSpecificDirectTransfer on the call [c ' A1] on Device A with ConsultCallID = CallID of [D ' A2] Direct Transfer / Join Across Line is not supported on CTI Remote Device	Line_Reply with error = LINEERR_OPERATIONUNAVAIL	
Variation: On any unsupported Feature Request For Example: CallAcceptRequest CallAnswerRequest CallParkRequest LineCallUnParkRequest	LINEERR_OPERATIONUNAVAIL Or PHONEERR_OPERATIONUNAVAIL Depending on the Line/Phone API request.	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

Cbarge

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A', A" and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
Make Call between C and A1[Remote Destinaton], either normal incoming or DVO call on CTI Remote Device Call Info is same above test cases		

Action	TAPI messages	TAPI structures
cBarge from CTI Remote Device is not supported as CTI Remote Device is a Static virtual Device. cBarge from EP [Enterprise phone] *** cBarge will be successful and CTIRemote Device, EP and Caller will be in Conference. *** as CTI Remote Device doesn't report RIU calls, there won't be RIU Conference created on CTI Remote Device reflecting Active Conference Call on EP	Conference Call model on CTI Remote Device : [C ' A1]-[Original Call1]-[state = Conference] [A1 ' Conference]-[Conference Parent Call]-[State = CONNECTED] [A1 ' A1(EP)]-[Consult Call]-[state -CONFERENCE] Call Model on Enterprise Phone: Active Conference Calls: [C ' A1(CTIRD)]-[Original Call1]-[state = Conference] [A1(EP) ' Conference]-[Conference Parent Call]-[State = CONNECTED] [A1(EP) ' A1(CTIRD)]-[Consult Call]-[state -CONFERENCE] RIU Conference Calls: [C ' A1]-[Original Call1]-[state = Conference] [A1 ' Conference]-[Conference Parent Call]-[State = CONNECTED] [A1 ' A1(EP)]-[Consult Call]-[state -CONFERENCE]	
Variation: Barge Operation on Enterprise Phone	Barge Operation will fail as CTI Remote Devices doesn't have BIB.	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

URI Dialing -Basic Incoming Call to CTI Remote Device

Precondition: InAddition to configuration from previous usecases

CTI Remote Device:

Line:

Line-A (DN -2000) (URI Configured -drajesh@cisco.com)

C (IP Phone -Name: SEPxxxxxxxx)

Line:

Line-C -DN -1000(URI configured -1000@cisco.com)

D (IP Phone -Name: SEPxxxxxxxx)

Line:

Line-D -DN -1001(URI configured -1001@cisco.com)

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A' and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	
LineMakeCall on Line-C with URI of CTI Remote Device (DestinationAddress -drajesh@cisco.com)	LineMakeCall() success Call on C : LINE_CALLSTATE -Param1 = DIALING LINE_CALLSTATE -Param1 = PROCEEDING LINE_CALLSTATE -Param1 = RINGBACK Call on CTI Remote Device : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Call on Enterprise Phone : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED	
After "Delay Before Ringing Timer" expires the call is offered on Remote Destinations and all Remote Destinations Ring		

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : Empty ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID =

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : Empty ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID =
Answer on any of the Remote Destination	Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED (active) Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = CTIRD-2000name SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID = 2000

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 1000 dwCallerIDName = 1000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: UnicodeCallerPartyName = 1000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = CTIRD-2000name SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] ModifiedCallingParty = 1000 ModifiedCalledParty = 2000 ModifiedConnectedID = 2000

Action	TAPI messages	TAPI structures
LineDrop() for the call on Device A (CTI-RD) Call on Remote Destination is dropped	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	
Variation : Answer the call on Enterprise Phone (B) LineAnswer() on the call on Device B Call on Remote Device/Remote Destination drops	Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED	

URI Dialing -DVO Call (Outgoing Call Initiation From CTI Remote Device)

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	
Open all Lines (A, A' and C) LineOpen() with ExtVer-0x000C0000	LineOpen() returns Success	

Action	TAPI messages	TAPI structures
LineMakeCall on Line-A with DN (C -DN 1000)	LineMakeCall() returns RequestID LINE_REPLY Param1 = RequestID Param2 = LINEERR_OPERATION_FAIL_NO_ACTIVE_RD_SET (0xC0000016)	
Update Remote Destination RD1 "Mobile"on CTI Remote Device A with IsActive set to true CiscoLineDevSpecific UpdateRemoteDestination Req m_RDNumber = "914086271309" m_UnicodeRDName = "Mobile" m_NewRDNumber = "914086271309" m_activeRD = 0x00000001	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_REMOTE_DESTINATION_INFO (0x00004000)	
LineGetDevCaps() with dwDeviceID = LineDeviceId of Line-A.	LineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific dwLineTypes = (0x00000000) DeviceProtocolType = DeviceProtocolType_CTI_ REMOTE_DEVICE (0x03) Remote Destination Info: unicodeRDName = "Mobile" RDNumber = "4086271309" isActiveRD = 0x00000001 unicodeRDName = "Office" RDNumber = "4089022131" isActiveRD = 0x00000000 IsMyAppLastToSetActiveRD = 0x00000001
LineMakeCall on Line-A with URI of C (DestinationAddress -1000@cisco.com) *** Only Remote Destination "Mobile" rings and it rings immediately as the RD is set Active *** No Call presented on EP	LineMakeCall() success Call on CTI Remote Device : LINE_CALLSTATE -Param1 = OFFERING	

Action	TAPI messages	TAPI structures
Answer the first Call on CTI Remote Device: Answer() on the call on CTIRemote Device(A)	LineAnswer() fail with Error LINEEE_OPERATIONUNAVAIL	
LineGetCallInfo() on call on Device A(CTIRD)	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 2000 dwCallerIDName = voiceConnect dwCalledID = 2000 dwCalledIDName = 2000name DevSpecific :: UnicodeCallerPartyName = UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = empty Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : [User | Host | Port | TransportType | URI Type] = empty ModifiedCallingParty = 2000 ModifiedCalledParty = 2000 ModifiedConnectedID =

Action	TAPI messages	TAPI structures
Once Remote Destination answers the call, call will be offered on initial dialed number C Call will be present on Enterprise Phone and call will be Remote In Use Call	Call on C : LINE_CALLSTATE -Param1 = OFFERING LINE_CALLSTATE -Param1 = ACCEPTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED LINE_CALLSTATE -Param1 = RINGBACK Call on Enterprise Phone : LINE_APPNEWCALL LINE_CALLSTATE -Param1 = ACCEPTED LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	
C answers the call LineAnswer() on call on Device-C	LineAnswer() success Call on C : LINE_CALLSTATE -Param1 = CONNECTED Call on CTI Remote Device : LINE_CALLSTATE -Param1 = CONNECTED (active) Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive)	

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device C	LineGetCallInfo() success	LineCallInfo :: CallReason = UNKNOWN (0x400) dwCallerID = 2000 dwCallerIDName = 2000name dwCalledID = 1000 dwCalledIDName = 1000name dwConnectedID = 2000 dwConnectedIDName = CTIRD-2000name DevSpecific :: ExtendedCallReason = CtiReasonMobility(0x021 = 33) UnicodeCallerPartyName = 2000name UnicodeCalledPartyName = 1000name UnicodeConnectedPartyName = 2000name SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [100 | Cisco.com | 0x0 | 0x0 | 0x1] Connected : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] ModifiedCallingParty = 2000 ModifiedCalledParty = 1000 ModifiedConnectedID = 2000

Action	TAPI messages	TAPI structures
LineGetCallInfo() on call on Device A/B	LineGetCallInfo() success	LineCallInfo :: dwCallerID = 2000 dwCallerIDName = 2000name dwCalledID = 2000 dwCalledIDName = 2000name dwConnectedID = 1000 dwConnectedIDName = 1000name DevSpecific :: CallAttributeType = TSPCallAttribute_DVOCall (0x00002000) UnicodeCallerPartyName = 2000name UnicodeCalledPartyName = 2000name UnicodeConnectedPartyName = 1000name SIP URI Info: Caller : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Called : [User | Host | Port | TransportType | URI Type] = [drajesh | Cisco.com | 0x0 | 0x0 | 0x1] Connected : [User | Host | Port | TransportType | URI Type] = [1000 | Cisco.com | 0x0 | 0x0 | 0x1] ModifiedCallingParty = 2000 ModifiedCalledParty = 2000 ModifiedConnectedID = 1000

Action	TAPI messages	TAPI structures
LineDrop() for the call on Device A (CTI-RD)	LineDrop() success Call on C : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on CTI Remote Device : LINE_CALLSTATE -Param1 = DISCONNECTED LINE_CALLSTATE -Param1 = IDLE Call on Enterprise Phone : LINE_CALLSTATE - Param1 = CONNECTED Param2 = 0x02 (Inactive) LINE_CALLSTATE -Param1 = IDLE	
Variation : Test the same with CSF Device in Extend Mode	Expected result would be same as observed on CTI Remote Device	

CTI RD Call Forwarding

Table 45: Use Case 1: Device A Calls CTIRD When Active RD Is Not Set and "Route calls to all remote destinations when client is not connected" Is Enabled.

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to CTIRD	Incoming calls are Forwarded to all remote destinations.

Table 46: Use Case 2: Device A Calls CTIRD When Active RD Is Not Set and "Route calls to all remote destinations when client is not connected" Is Disabled. There Is No Call Forward Number Set on the Shared Enterprise Phone

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to CTIRD	Call is disconnected with reason code -USER_BUSY.

Table 47: Use Case 3: Device 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 (CFNA Is Configured On Enterprise Number to Voice Mail Box)

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to CTIRD	Call will route to voice mail number.

Table 48: Use Case 4: Device 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 (CFNA Is Configured On Enterprise Number to Voice Mail Box)

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to CTIRD	Call will route to voice mail number.

Table 49: Use Case 5: DeviceA Calls CTIRD When Active RD Is Set and "Route calls to all remote destinations when client is not connected" Is Enabled. Setup: A IP Phone, B CTI-RD, C RDD1, D RDD2. Active RD Is Set to C

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to B 4. C answers the call	Incoming calls is routed to active remote destination, such as C.

Table 50: Use Case 6: Device A Calls CTIRD When Active RD Is Set and "Route calls to all remote destinations when client is not connected" Is Enabled. Setup: A IP Phone, B CTI-RD, C RDD1, D RDD2. Active RD Is Set to C

Scenario	Expected Result
1. Provider Open request 2. Issue Line Open on remote device and devices which have the remote destinations 3. Phone A makes a call to B	Incoming calls is routed to active remote destination.

Video Capabilities and Multimedia Information

Use cases related to Video Capabilities and Multi-Media Information feature are mentioned below:

Media Capability on Device A (SIP Phone with Camera) Which Is Video-Enabled, Supports Telepresence, and Has 2 Screens

Action	Expected events
LineInitializeEx Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Video Capability = 0x00000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 1 ScreenCount = 2

Media Capability on Device A (SIP Phone) Which Is Not Video-Enabled, Supports Telepresence, and Has 2 Screens

Action	Expected events
LineInitializeEx Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Media Capability = 0x00000000 [CiscoDeviceVideoCapability_None] TelepresenceInfo = 1 ScreenCount = 2

Media Capability on Device A (CTI Port/Remote Point)

Action	Expected events
LineInitializeEx Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Media Capability = 0x00000000 [CiscoDeviceVideoCapability_None] TelepresenceInfo = 0 Screen Count = 0

Media Capability on an Acquired Device B Which Is Media-Enabled (super Provider Scenario), Supports Telepresence, and Has 3 Screens

Action	Expected events
LineInitializeEx LineOpen with Ext version 0x000D0000 with deviceId for linedevice A Issue CCiscoLineDevSpecificAcquire to Acquire Device B. Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice B LineShutdown	LineOpen successful. Device Acquired Successfully. LINE_CREATE message fired. LINEGETDEVCAPS::DEVSPECIFIC exposes Media Capability = 0x00000001 [CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 1 Screen Count = 3

Media Capability on Device A (ParkDN/Pickupdevice)

Action	Expected events
LineInitializeEx Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Media Capability = 0x00000000 [CiscoDeviceVideoCapability_None] TelepresenceInfo = 0 Screen Count = 0

Media Capability on Device A (SIP Phone Which Is Unregistered and Is Video-Enabled)

Action	Expected events
LineInitializeEx Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Media Capability = 0x00000000 [CiscoDeviceVideoCapability_None] TelepresenceInfo = 0 Screen Count = 0

Video Capability on Device B (A Is a SIP Phone with Video-Enabled and B Is SIP Phone with Video-Enabled) , Both Devices Support Telepresence, and Have 3 Screens

Action	Expected events
LineInitializeEx A does a LineMakeCall to B, B answers. Issue LineGetcallInfo() with Ext version for linedevice B LineShutdown	B : LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapabilities : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3 CalledPartyVideoCapabilities : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3

Action	Expected events
Variation 1: A has video enabled and B has video disabled. A has Telepresence enabled and has 3 screens, B has Telepresence disabled and has 1 screens.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapabilities : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3 CalledPartyVideoCapabilities : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 1
Variation 2: A has video enabled,1 scren and B is a CTI Port or Route Point.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapabilities : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 1 CalledPartyVideoCapabilities : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 0

Video Capability on Device C After Redirect (A Is a SIP Phone Which Is Video-Disabled, B and C Are Video-Enabled)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B redirects to C, C answers Issue LineGetcallInfo() with Ext version for linedevice C LineShutdown	C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled

Video Capability on Device C After Blindtransfer (A Is a SIP Phone Which Is Video-Disabled, B and C Are Video-Enabled)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B does a blindtransfers to C, C answers Issue LineGetcallInfo() with Ext version for linedevice C LineShutdown	C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled

Video Capability on Device C After Consult Transfer (A Is a SIP Phone Which Is Video-Disabled, B and C Are Video-Enabled)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B does a LineSetupTransfer to C, C answers B does a LineCompleteTransfer Issue LineGetcallInfo() with Ext version for linedevice C LineShutdown	C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled

Video Capability on Device B on an Existing Call (Both A and B Are SIP Phones Which Are Video-Enabled)

Action	Expected events
A does a Call to B, B answers. LineInitializeEx Issue LineGetcallInfo() with Ext version for linedevice B LineShutdown	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
Variation 1: A has video enabled and B has video disabled.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None

Action	Expected events
Variation 2: A has video enabled and B is a CTI Port or Route Point.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None

Dynamic Media Capability Change on Device A (SIP Phone with Camera) Which Is Video-Enabled

Action	Expected events
LineInitializeEx LineOpen on A Issue LineGetDevCaps() with Ext version 0x000D0000 with deviceId for linedevice A Change Video Capability of device to Disabled from CUCM Admin page LineShutdown	LINEGETDEVCAPS::DEVSPECIFIC exposes Video Capability = 0x00000001[CiscoDeviceVideoCapability_Enabled] TSP will fire SLDSMT_LINE_PROPERTY_CHANGED event to application with dwParam2 = LPCT_DEVICE_VIDEO_INFO(0x00010000).
Variation 1: Intially Device A has Video disabled and then change Video Capability of device to enabled from CUCM Admin page.	TSP will fire SLDSMT_LINE_PROPERTY_CHANGED event to application with dwParam2 = LPCT_DEVICE_VIDEO_INFO(0x00010000).

Video Capability on Device A and B; Both Are Video-Enabled SIP Phones And, Both Devices Support Telepresence and Has 3 Screens

Action	Expected events
LineInitializeEx LineOpen on A and B A does a LineMakeCall to B, B answers. Issue LineGetcallInfo() with Ext version for linedevice A LineShutdown	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapBitMask = 0x00000007 CalledPartyMultiMediaCapBitMask = 0x00000007 CallingPartyMultiMediaCapInfo : VideoCapability = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3 CalledPartyMultiMediaCapInfo : VideoCapability = 0x00000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x00000001(Telepresence Enabled Screen Count = 3
Variation 1: A has video enabled and B has video disabled. A has Telepresence enabled and has 3 screens, B has Telepresence disabled and has 1 screens.	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapBitMask = 0x00000007 CalledPartyMultiMediaCapBitMask = 0x00000007 CallingPartyMultiMediaCapInfo : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3 CalledPartyMultiMediaCapInfo : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 1

Action	Expected events
Variation 2: A has video enabled,1 screen and B is a CTI Port or Route Point.	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapBitMask = 0x00000007 CalledPartyMultiMediaCapBitMask = 0x00000000 CallingPartyMultiMediaCapInfo : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 0x00000001 CalledPartyMultiMediaCapInfo : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 0x00000000

Check If the Multimedia Streams Info Has Not Returned on the Call From Both Calling Party and Called Party, If Lines Are Opened with Ext 0x000B0000 (TLS Connections Must Be Disabled, Phone A and B Are Video-Disabled)

Action	Expected events
LineInitializeEx LineOpen at A and B with extension version 0x000B0000 A does a LineMakeCall to B / B answers the call Check there is no CallDevSpecific event returned.	No CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA

Check If the Multimedia Streams Info Has Returned on the Call From Both Calling Party and Called Party, If Lines Are Opened with Ext 0x000D0000 (TLS Connections Must Be Disabled, Phone A and B Are Video-Enabled)

Action	Expected events
LineInitializeEx LineOpen at A and B with extension version 0x000B0000 A does a LineMakeCall to B / B answers the call Check there is CallDevSpecific event returned. LineGetCallInfo on A	

Action	Expected events
	CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA DevSpecificPart of LINECALLINFO For Party A: Video Stream Information returned for the following: CompressionType = The actual compression type BitRate = The actual bit rate MediaMode = 0x00000000 PacketSize = The actual packet size bSilenceSupressionFlag = 0x00000000 bKeyInfoPresen = 0x00000000 RxRTPDestinationV6Offset = The actual IPV6 address offset RxRTPDestinationV6Size = The actual IPV6 address size RxRTPIPV4Address = The actual IPV4 address RxRTPIPV4Por t = The actual IPV4 port RxIpAddrMode = The actual IPV4 mode TxRTPDestinationV6Offset = The actual IPV6 address offset TxRTPDestinationV6Size = The actual IPV6 address size TxRTPIPV4Address = The actual IPV4 address TxRTPIPV4Port = The actual IPV4 port TxIpAddrMode = The actual IPV4 mode MultiMediaEncryptionKey Information returned is the following AlgorithmID = 0x00000000 TxKeyOffset = 0x00000000 TxKeySize = The actual size RxKeyOffset = The actual offset RxKeySize = The actual size TxSaltOffset = The actual offset TxSaltSize = The actual size RxSaltOffset = The actual offset RxSaltSize = The actual size TxIsMKIPresent = 0x00000000 RxIsMKIPresent = 0x00000000 SecurityIndicator = 0x00000001

Action	Expected events
Variation 1: A does a LineMakeCall to B / B answers the call Application does LineHold on B LineGetCallInfo on A and B Application does LineUnHold on B LineGetCallInfo on A and B Application does a LineDrop on B. LineGetCallInfo on A and B	CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000003 CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000000 CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000003

Negotiated Video Capability Will Be Reported to the Called Party Accross a Inter Cluster Call (over SIP – ICT Trunk) Using Early Offer (Phone A Is Video-Disabled SIP Phone and Phone B Is Video-Enabled, A Is in Cluster 1 and B Is in Cluster 2)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B answers. LineGetCallInfo on A LineGetCallInfo on B LineShutdown	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

Action	Expected events
Variation 1: A and B are SIP Phone and have video enabled. LineGetCallInfo on A LineGetCallInfo on B	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

Multiple Redirect Over SIP Trunk (Phone A, B, and C Are Video-Enabled SIP Phones, Phone D Is Video-Disabled. Phone A Is in Cluster 1 and Phone B, C, and D Are in Cluster 2)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. LineGetCallInfo on B B redirects the call to C, LineGetCallInfo on C C redirects the call to D, LineGetCallInfo on D LineShutdown	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] D: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled]

Redirect Over SIP Trunk (Phone A Is Video-Enabled SIP Phone and Phone B and C Is Video-Disabled, Phone A Is in Cluster 1 and Phone B and C Are in Cluster 2)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B answers. B redirects to C, C answers. LineGetCallInfo on A LineGetCallInfo on C LineShutdown A and B have video enabled, C has video disabled A does a LineMakeCall to B. B answers. B redirects to C, C answers. LineGetCallInfo on A LineGetCallInfo on C	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled]

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)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B answers. B Holds the call. C Unholds the call. LineGetCallInfo on A LineGetCallInfo on C LineShutdown A and B are have video enabled and C has video disabled. A does a LineMakeCall to B. B answers. B Holds the call. C Unholds the call. LineGetCallInfo on A LineGetCallInfo on C	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled]

Multiple Redirect Over H323 ICT Trunk (Phone A, B, C and D Are Video-Enabled SIP Phones, Phone A Is in Cluster 1 and Phone B, C, and D Are in Cluster 2)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. LineGetCallInfo on B B redirects the call to C. LineGetCallInfo on C C redirects the call to D. LineGetCallInfo on D LineShutdown	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] D: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

Redirect Over H323 Trunk (Phone A Is Video-Enabled SIP Phone and Phone B and C Are Video-Disabled, Phone A Is in Cluster 1 and Phone B and C Are in Cluster 2)

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B answers. B redirects to C, C answers. LineGetCallInfo on A LineGetCallInfo on C LineShutdown A and B have video enabled, C has video disabled A does a LineMakeCall to B. B answers. B redirects to C, C answers. LineGetCallInfo on A LineGetCallInfo on C	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

Direct Transfer Across Lines

Use cases related to Direct Transfer Across Lines feature are mentioned below:

Note The device mentioned in the use cases also apply to SCCP device and SIP TNP phones when Direct Transfer is issued from application.

Direct Transfer Across Lines on RoundTable Phones via Application

Device A, B, and C where B is roundtable phone and has line B1 and B2 configured.

Action	Expected events
A ‡B1 is connected, C ‡B2 is on hold	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1 For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1, Connected = A For B2: LINE_CALLSTATE param1 = x100, HOLD Caller = C, Called = B2 , Connected = C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
Application sends CciscoLineDevSpecificDirectTransfer on B1 with B2 as consult call	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected C For B1: Call goes IDLE For B2: Call goes IDLE For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = A

Direct Transfer on Same Line on RoundTable Phones Via Application

Device A, B, C where B is roundtable phone.

Action	Expected events
A ‡ B (c1) is connected, C ‡ B (c2) is on hold	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B For B: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A Call-2 LINE_CALLSTATE param1 = x100, HOLD Caller = C, Called = B, Connected = C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B, Connected = B
Application sends CciscoLineDevSpecificDirectTransfer on B (c1) with c2 as consult call	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For B: Call-1 and Call-2 will go IDLE For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B, Connected = A

Direct Transfer Across Lines on RoundTable Phones via Application with Call in Offering State

Device A, B, C where B is roundtable phone and has line B1 and B2 configured.

Action	Expected events
A (c1) ‡ B1(c2) is on hold, B2 (c3) ‡ C (c4) is ringing	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1 For B1: LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = B1, Connected = A For B2: LINE_CALLSTATE param1 = x100, RINGBACK Caller = B2, Called = C For C: LINE_CALLSTATE param1 = x100, OFFERING Caller = B2, Called = C
Application sends CciscoLineDevSpecificDirectTransfer on B1 (c2) with B2 (c3) as consult call	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For B1: Call goES IDLE For B2: Call goes IDLE For C: LINE_CALLSTATE param1 = x100, OFFERING Caller = C, Called = B,

Failure of Direct Transfer Calls Across Lines

Device A, B, C where B is roundtable phone and has line B1 and B2 configured.

Action	Expected events
A (c1) ‡ B1(c2) is on hold, Initiate new call (c3) on B2	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1 For B1: LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = B1, Connected = A For B2: LINE_CALLSTATE param1 = x100, DIALTONE
Application sends CciscoLineDevSpecificDirectTransfer on B1 (c2) with B2 (c3) as consult call	CciscoLineDevSpecificDirectTransfer gets error as LINEERR_INVALCALLSTATE.

Direct Transfer Calls Across Lines in Conference Scenario

Device A, B, C, D and E where C is roundtable phone and has line C1 and C2 configured.

Action	Expected events
A/B/C1 in conference, B is controller, call on C1 is in hold state. C2 /D/E in conference, D is controller, call on C2 is in connect state.	For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = C1, connected = C1
	For B: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = C1, connected = C1
	For C1: ONHOLD CONFERENCED Caller = B, called = C1, connected = B CONFERENCED Caller = C1, called = A, connected = A
	For C2: CONNECTED CONFERENCED Caller = C2, called = D, connected = D CONFERENCED Caller = C2, called = E, connected = E
	For D: CONNECTED CONFERENCED Caller = D, called = C1, connected = C1 CONFERENCED Caller = D, called = E, connected = E

Action	Expected events
	For E: CONNECTED CONFERENCED Caller = D, called = E, connected = D CONFERENCED Caller = E, called = C2, connected = C2

Action	Expected events
Application sends CciscoLineDevSpecificDirectTransfer on C1 with C2-call as consult call	CciscoLineDevSpecificDirectTransfer will succeed. For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = CB-2, connected = CB-2 For B: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = CB-2, connected = CB-2 For C1: IDLE For C2: IDLE For D: CONNECTED CONFERENCED Caller = D, called = CB-1, connected = CB-1 CONFERENCED Caller = D, called = E, connected = E For E: CONNECTED CONFERENCED Caller = D, called = E, connected = D CONFERENCED Caller = E, called = CB-1, connected = CB-1

Connect Transfer Across Lines on RoundTable Phones

Device A, B, C where B is roundtable phone and has line B1 and B2 configured.

Action	Expected events
A ‡ B1 is connected, C ‡ B2 is on hold	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1 For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1, Connected = A For B2: LINE_CALLSTATE param1 = x100, HOLD Caller = C, Called = B2, Connected = C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
User performs connect transfer on B.	For A: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected C For B1: Call goes IDLE For B2: Call goes IDLE For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = A

Do Not Disturb-Reject

Application Enables DND-R on a Phone

Action	TAPI messages	TAPI structures
Phone A enables DND-Reject in the admin pages	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_DND_OPTION_STATUS dwParam3 = 2	

Normal Feature Priority

Action	TAPI messages	TAPI structures
With Phone B DND-R enabled, Phone A calls Phone B with feature priority as Normal	Party A	
	LINE_CALLSTATE = IDLE	
	Party B	
	No TAPI messages	

Feature Priority - Emergency

Action	TAPI messages	TAPI structures
With Phone B DND-R enabled, Phone A calls Phone B with feature priority as Emergency	Party A	

Action	TAPI messages	TAPI structures
	LINE_CALLSTATE = CONNECTED dwParam1 = 0x00000100 dwParam2 = 0x00000001	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwCallerID = A dwCalledID = B dwRedirectionID = NP dwRedirectingID = NP
	Party B	
	LINE_CALLSTATE = CONNECTED dwParam1 = 0x00000100 dwParam2 = 0x00000001	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwCallerID = A dwCalledID = B dwRedirectionID = NP dwRedirectingID = NP

Shared Line Scenario for DND-R

Action	TAPI messages	TAPI structures
Phones B and B' represents shared lines. Phone B' is DND-R enabled but not B. Phone A calls Phone B with feature priority normal	Party A	
	LINE_CALLSTATE = CONNECTED dwParam1 = 0x00000100 dwParam2 = 0x00000001	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwCallerID = A dwCalledID = B dwRedirectionID = NP dwRedirectingID = NP
	Party B	

Action	TAPI messages	TAPI structures
	LINE_CALLSTATE = CONNECTED dwParam1 = 0x00000100 dwParam2 = 0x00000001	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwCallerID = A dwCalledID = B dwRedirectionID = NP dwRedirectingID = NP
	Party B’	
	LINE_CALLSTATE = CONNECTED dwParam1 = 0x00000100 dwParam2 = 0x00000002	

Application Disables DND-R or Changes the Option for DND

Action	TAPI messages	TAPI structures
Phone A changes from DND-Reject to DND-RingerOff.	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_DND_OPTION_STATUS dwParam3 = 1	

Drop Any Party

Use cases related to Drop Any Party feature are mentioned below:

Conference: Unified CM Service Parameter Advanced Ad Hoc Conference Enabled = False

Action	Expected events
A,B,C and D are in conference; B is conference Controller.	Conference Model: Each line in conference will be having 4 callLegs, 3 conferenced and 1 connected
	CallLegs on A: Connected -to Conference Bridge Conferenced -(Connected Id -B) Conferenced -(Connected Id -C) Conferenced -(Connected Id -D)
	CallLegs on B: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -C) Conferenced -(Connected Id -D)
	CallLegs on C: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -D)
	CallLegs on D: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -C)
Application does a LineOpen (B) with new Ext ver.	

Action	Expected events
1. Application does LineRemoveFromConference on the 'Conferenced' callLeg on B which is connected to A.	A is dropped out of conference. CallLegs after the Party is dropped from Conference: Each line in conference will be having 4 callLegs, 2 Conferenced,1 IDLE and 1 connected
	CallLegs on A: All 4 CallLegs will be in IDLE state
	CallLegs on B: Connected -to Conference Bridge Conferenced -(Connected Id -C) Conferenced -(Connected Id -D) IDLE -(on the conferenced callLeg which was connected to A)
	CallLegs on C: Connected -to Conference Bridge IDLE -(on the conferenced callLeg which was connected to A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -D)
	CallLegs on D: Connected -to Conference Bridge IDLE -(on the conferenced callLeg which was connected to A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -C) **Note** All IDLE CallLegs will have CallStateChange Reason as CtiDropConferee.
Application does a LineOpen (A) with new Ext ver.	
1. Application does LineRemoveFromConference on the 'Conferenced' callLeg on A which is connected to B.	Error Message LINEERR_OPERATIONUNAVAIL will be sent to application

Conference: Unified CM Service Parameter Advanced Ad Hoc Conference Enabled = True

Action	Expected events
A,B,C and D are in conference; B is conference Controller.	Conference Model: Each line in conference will be having 4 callLegs, 3 conferenced and 1 connected
	CallLegs on A: Connected -to Conference Bridge Conferenced -(Connected Id -B) Conferenced -(Connected Id -C) Conferenced -(Connected Id -D)
	CallLegs on B: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -C) Conferenced -(Connected Id -D)
	CallLegs on C: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -D)
	CallLegs on D: Connected -to Conference Bridge Conferenced -(Connected Id -A) Conferenced -(Connected Id -B) Conferenced -(Connected Id -C)
Application does a **LineOpen (A)** with new Ext ver. Application does LineRemoveFromConference on the 'Conferenced' callLeg on A which is connected to B.	

Action	Expected events
1. Drop Ad Hoc Conference = Never	B is dropped out of conference. **CallLegs after the Party is dropped from Conference**: Each line in conference will be having 4 callLegs, 2 Conferenced,1 IDLE and 1 connected
	CallLegs on B: All 4 CallLegs will be in IDLE state
	CallLegs on A: Connected -to Conference Bridge Conferenced -(Connected Id -C) Conferenced -(Connected Id -D) IDLE -(on the conferenced callLeg which was connected to B)
	CallLegs on C: Connected -to Conference Bridge IDLE -(on the conferenced callLeg which was connected to B) Conferenced -(Connected Id -A) Conferenced -(Connected Id -D)
	CallLegs on D: Connected -to Conference Bridge IDLE -(on the conferenced callLeg which was connected to B) Conferenced -(Connected Id -A) Conferenced -(Connected Id -C) **Note** All IDLE CallLegs will have CallStateChange Reason as CtiDropConferee.
1. Drop Ad Hoc Conference = 'When Conference Controller Leaves'	B is dropped out of conference and Conference will be ended. **CallLegs after the Party is dropped from Conference**: Each line in conference will be having 4 callLegs, all in IDLE state **CallLegs on A,B,C and D**: All 4 CallLegs will be in IDLE state

Shared Line-Scenario

Action	Expected events
A,B,C and A' are in conference; A is conference Controller Unified CM Parameter "Drop Ad Hoc Conference = Never"	**Conference Model**: Lines B and C in conference will be having 4 callLegs, 3 conferenced and 1 connected Lines A and A' will be having 8 CallLegs
	CallLegs on A: Connected -to Conference Bridge (Active) Conferenced -(caller Id -A ;Called Id -B; Connected Id -B) (Active) Conferenced -(caller Id -A ;Called Id -C; Connected Id -C) (Active) Conferenced -(caller Id -A ;Called Id -A' ; Connected Id -A') (Active) Connected -to Conference Bridge (Remote in Use) Conferenced -(caller Id -A' ;Called Id -B; Connected Id -B) (Remote in Use) Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Remote in Use) Conferenced -(caller Id -A' ;Called Id -A; Connected Id -A) (Remote in Use)

Action	Expected events
	CallLegs on A': Connected -to Conference Bridge (Active) Conferenced -(caller Id -A' ;Called Id -B; Connected Id -B) (Active) Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Active) Conferenced -(caller Id -A' ;Called Id -A; Connected Id -A) (Active) Connected -to Conference Bridge (Remote in Use) Conferenced -(caller Id -A ;Called Id -B; Connected Id -B) (Remote in Use) Conferenced -(caller Id -A ;Called Id -C; Connected Id -C) (Remote in Use) Conferenced -(caller Id -A ;Called Id -A'; Connected Id -A') (Remote in Use) **CallLegs on B**: Connected -to Conference Bridge Conferenced -(caller Id -B ;Called Id -A; Connected Id -A) Conferenced -(caller Id -B ;Called Id -C; Connected Id -C) Conferenced -(caller Id -B ;Called Id -A'; Connected Id -A') **CallLegs on C**: Connected -to Conference Bridge Conferenced -(caller Id -C ;Called Id -A; Connected Id -A) Conferenced -(caller Id -C ;Called Id -B; Connected Id -B) Conferenced -(caller Id -C ;Called Id -A' ; Connected Id -A')
Application does a **LineOpen (A)** with new Ext ver. Unified CM Parameter '**Advanced Ad Hoc Conference Enabled = False**'	
1. Application does **LineRemoveFromConference** on the 'Conferenced' CallLeg on A which is connected to B and mode is "Inactive or Remote In use".	Error LINEERR_INVALCALLSTATE is sent to application.

Action	Expected events
1. Application does **LineRemoveFromConference** on the 'Conferenced' CallLeg on A which is connected to B and mode is 'Active'.	B will be dropped out of conference. LINECALLSTATE Event will be sent to Application with state = Idle.

Action	Expected events
	CallLegs after the Party is dropped from Conference: **CallLegs on A**: Connected -to Conference Bridge (Active) IDLE -(on the conferenced callLeg which was connected to A -B) Conferenced -(caller Id -A ;Called Id -C; Connected Id -C) (Active) Conferenced -(caller Id -A ;Called Id -A'; Connected Id -A') (Active) Connected -to Conference Bridge (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A' -B) Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Remote in Use) Conferenced -(caller Id -A' ;Called Id -A; Connected Id -A) (Remote in Use) **CallLegs on A'**: Connected -to Conference Bridge (Active) IDLE -(on the conferenced callLeg which was connected to A' -B) Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Active) Conferenced -(caller Id -A' ;Called Id -A; Connected Id -A) (Active) Connected -to Conference Bridge (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A -B) Conferenced -(caller Id -A ;Called Id -C; Connected Id -C) (Remote in Use) Conferenced -(caller Id -A ;Called Id -A'; Connected Id -A') (Remote in Use) **CallLegs on B:** All 4 CallLegs are in IDLE state **CallLegs on C**: Connected -to Conference Bridge Conferenced -(caller Id -C ;Called Id -A; Connected Id -A) IDLE -(on the conferenced callLeg which was connected to C

Action	Expected events
	-B) Conferenced -(caller Id -C ;Called Id -A'; Connected Id -A')
Application does a **LineOpen (B)** with new Ext ver. Unified CM Parameter **Advanced Ad Hoc Conference Enabled = True**	

Action	Expected events
1. Application does **LineRemoveFromConference** on the 'Conferenced' CallLeg on B which is connected to A and mode is "Active".	A will be dropped out of conference. LINECALLSTATE Event will be sent to Application with state = Idle.
	CallLegs after the Party is dropped from Conference: **CallLegs on A**: IDLE -(on the Connected callLeg which was connected to Conference Bridge,A-CFB) IDLE -(on the conferenced callLeg which is connected to A -B) IDLE -(on the conferenced callLeg which is connected to A -C) IDLE -(on the conferenced callLeg which is connected to A -A') Connected -to Conference Bridge (Remote in Use) Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Remote in Use) Conferenced -(caller Id -A' ;Called Id -B; Connected Id -B) (Remote in Use)
	CallLegs on A': IDLE -(on the Connected callLeg which was connected to Conference Bridge,A -CFB) IDLE -(on the conferenced callLeg which is connected to A -B) IDLE -(on the conferenced callLeg which is connected to A -C) IDLE -(on the conferenced callLeg which is connected to A -A') Connected -to Conference Bridge Conferenced -(caller Id -A' ;Called Id -C; Connected Id -C) (Active) Conferenced -(caller Id -A' ;Called Id -B; Connected Id -B) (Active)
	CallLegs on B: Connected -to Conference Bridge Conferenced -(caller Id -B ;Called Id -A; Connected Id -A') IDLE -(on the conferenced callLeg which was connected to B -A) Conferenced -(caller Id -B ;Called Id -C; Connected Id -C)

Action	Expected events
	CallLegs on C: Connected -to Conference Bridge Conferenced -(caller Id -C ;Called Id -A'; Connected Id -A') IDLE -(on the conferenced callLeg which was connected to C -A) Conferenced -(caller Id -C ;Called Id -B; Connected Id -B)

Chained Conference

Action	Expected events
A,B and CB2 are in conference(CB1); B is conference Controller C,D and E are in Conference (CB2); D is conference Controller Unified CM Parameter Advanced Ad Hoc Conference Enabled = True Application does a **LineOpen (A)** with new Ext ver. **1.** Application does **LineRemoveFromConference** on the Conferenced" CallLeg on A which is connected to B.	B is disconnected and dropped out of Conference. A is now in conference with CB2. LINECALLSTATE Event is sent to Application for Line B with state = Idle.

C-Barge: Unified CM Service Parameter Advanced Ad Hoc Conference Enabled = True.

Action	Expected events
B call A and A'; A answers the call and on A' do c-Barge; A,B and A' will be in conference; A is conference Controller Unified CM Parameter "Drop Ad Hoc Conference = Never" Application does a LineOpen (A) with new Ext ver.	

Action	Expected events
Application does a LineOpen (A) with new Ext ver. 1. Application does **LineRemoveFromConference** on the "Conferenced" CallLeg on A which is connected to B and mode is Active	B is dropped out of conference. LINECALLSTATE Event will be sent to Application with state = Idle. **CallLegs after the Party is dropped from Conference**: **CallLegs on A**: Connected -(on the conferenced callLeg which was connected to A -A') (Active) Connected -on the conferenced callLeg which was connected to A' -A) (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A -B) IDLE -(on the connected callLeg which is connected to conference Bridge; A -CFB) IDLE -(on the conferenced callLeg which was connected to A' -B) IDLE -(on the connected callLeg which is connected to conference Bridge; A' -CFB) **CallLegs on A'**: Connected -(on the conferenced callLeg which was connected to A' -A) (Active) Connected -on the conferenced callLeg which was connected to A -A') (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A -B) IDLE -(on the connected callLeg which is connected to conference Bridge; A -CFB) IDLE -(on the conferenced callLeg which was connected to A' -B) IDLE -(on the connected callLeg which is connected to conference Bridge; A' -CFB) **CallLegs on B**: All 4 CallLegs are in IDLE state A' is dropped out of conference. LINECALLSTATE Event will be sent to Application with state = Idle.

Action	Expected events
1. Application does LineRemoveFromConference on the Conferenced CallLeg on A which is connected to A' and mode is Active.	**CallLegs on A'**: Connected -(on the conferenced callLeg which was connected to A -B) (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A' -B) IDLE -(on the conferenced callLeg which was connected to A -A') (active) IDLE -(on the connected callLeg which is connected to conference Bridge; A -CFB) IDLE -(on the conferenced callLeg which was connected to A' -A) (Remote in Use) IDLE -(on the connected callLeg which is connected to conference Bridge; A' -CFB) **CallLegs on B**: Connected -(on the conferenced callLeg which was connected to B -A) IDLE -(on the conferenced callLeg which was connected to A' -B) IDLE -(on the connected callLeg which is connected to conference Bridge; B -CFB) **CallLegs after the Party is dropped from Conference**: **CallLegs on A**: Connected -(on the conferenced callLeg which was connected to A -B) (Active) IDLE -(on the conferenced callLeg which was connected to A' -B) (Remote in Use) IDLE -(on the conferenced callLeg which was connected to A -A') (active) IDLE -(on the connected callLeg which is connected to conference Bridge; A -CFB) IDLE -(on the conferenced callLeg which was connected to A' -A) (Remote in Use) IDLE -(on the connected callLeg which is connected to conference Bridge; A' -CFB)

Early Offer

The following section describes how the application dynamically registers for various port with Early Offer Support.

Application Dynamically Registers CTI Port with Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState dwParam1 = x040, InService
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success

Action	TSP message to application data
Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 1 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) – IPAddressing Mode
Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info	Line_Reply with Success
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) – IPAddressing Mode
Hold and unHold the Call	A: LINE_CALLSTATE (LINECALLSTATE_HOLD/LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 1 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) – IPAddressing Mode *** Applications have to set the RTP info as the SetRTP flag is set.
Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info	Line_Reply with Success Media will be set and Media events will be reported

Action	TSP message to application data
*** Application should not set the RTP Info Again Variant 1: Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info	Line_Reply with Error LINEERR_OPERATIONUNAVAIL But the Media is setup with the RTP information provided at the SLDSMT_RTP_GET_IP_PORT information request
Variant 2: Application does not set the Filter to receive new Notification using lineDevSpecific (CCiscoLineDevSpecificSetStatusMsgs) and Application does not Set RTP at Proceeding State as there is no Notification Or Application does not set RTP info on New Notification	New Notification not reported to Application Call goes to Disconnect State with cause as LINEDISCONNECTMODE_UNKNOWN
Variant 3: A – CTI Port is Registered Secure	Behavior should be same
Variant 4: Application tried to disable the Early Offer support on the CTI Port that is Dynamically Registered with the Early Offer support Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability -0x00000000	Line_Devspecific fails with Error LINEERR_OPERATIONUNAVAIL

Application Dynamically Registers CTI Port Without Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Delayed Offer

Action	TSP message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success

Action	TSP message to application data
Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService
Application calls LineMakeCall() on A dialing a Party in Cluster2	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 1 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) -IPAddressingMode
Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Inf0	Line_Reply with Success Media will be Setup
Variant 1: A – SCCP/SIP Phone	Behavior is same and new SLDSMT_RTP_GET_IP_PORT Notification will not be fired to application.

Application Dynamically Registers IPV6 CTI Port with Early Offer Support

Configuration

A – CTI Port; CDC – IPV6 Only

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success

Action	TSP message to application data
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
Application sends lineDevSpecific(CciscoLineDevSpecificSetIPv6AddressAndMode) with MediaCaps Info Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info	Line_Reply with Success Line_Reply with Success LineInserviceEvent will be repored to Application Line_LineDevState Dwparam1 = x040, InService
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is routed through SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) **Note** SLDSMT_RTP_GET_IP_PORT Notification for IPV6 CTI Port is not supported. Application has to set the RTP info after OpenLogicalChannel Notification.
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 1 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits)-IPAddressingMode
Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCallIPv6) with IPAddress and Port Info	Line_Reply with Success Media will be Setup

Mutiple Applications Dynamically Register CTI Port/RP

Configuration

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

- App1 – Dynamically Registers CTI Port/RP with Early Offer Support
- App2 – Dynamically Registers CTI Port/RP without Early Offer Support

*** App1 and App2 are running on Different Client Machines.

Action	TSP message to application data
App1 and App2: lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
App1 and App2: lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
App1 and App2: LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
App1: Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
App1: Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to the application.
App2: Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info	Line_Devspecific fails with Error LINEERR_REGISTER_GETPORT_SUPPORT_MISMATCH

Multiple Applications Dynamically Register CTI Port/RP with Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

- App1 – Dynamically Registers CTI Port/RP with Early Offer Support
- App2 – Dynamically Registers CTI Port/RP with Early Offer Support

*** App1 and App2 are running on Different Client Machines.

Action	TSP Message to application data
App1 and App2: lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
App1 and App2: lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
App1 and App2: LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
App1 and App2: Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
App1 and App2: Application sends lineDevSpecific(CciscoLineDevSpecificPortRegistrationPerCall) with MediaCaps Info *** Both Applications set with same Capabilities	Line_Reply with Success LineInserviceEvent reports to Application.
App1: Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) App1 and App2: LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 1 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) uy (8 bits) – IPAddressing Mode
App1: Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info	Line_Reply with Success
App2: Application sends LineDevSpecific (CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info different from the Info App1 has set.	Line_Reply with error LINEERR_OPERATIONUNAVAIL

Action	TSP Message to application data
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) – IPAddressingMode

Application Statically Registers CTI Port with Early Offer Support and Then Disable the Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState dwParam1 = x040, InService
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success

Action	TSP Message to application data
Application calls LineMakeCall() on A dialing a Party in Cluster 2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)
*** Disconnect the Existing Call Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability -0x00000000 – to disable the Early Offer support	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster 2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING/ LINECALLSTATE_RINGBACK)
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)

Application Statically Registers CTI Port with Out Early Offer Support and Then Enables Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success

Action	TSP Message to application data
Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001 – to enable the Early Offer support	Line_Reply with Success
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) Media will be set and Media Events will be Reported to Application
Variant 1: A – SCCP/SIP Phone	Behavior is same and new SLDSMT_RTP_GET_IP_PORT Notification will not be fired to application.

Application Registers CTI Port with Legacy Wave Driver and Enables Early Offer Support

Configuration

A – CTI Port;

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x000B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Devspecific fails with error LINEERR_OPERATIONUNAVAIL
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is routed through SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) Media will be set and Media Events will be reported to Application

Application Registers CTI Port with New Cisco Wave Driver and Enables Early Offer Support

Configuration

A – CTI Port;

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
During Installation of CiscoTSP User has to select New Wave Driver. lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x000B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService

Action	TSP Message to application data
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is routed through SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode **Note** On this new Notification, applications has to Open the Port.
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) Media will be set and Media Events will be reported to Application

Mutiple Applications Statically Register CTI Port

Configuration

A – CTI Port in Cluster 1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

- App1 – Statically Registers CTI Port/RP with Early Offer Support
- App2 – Statically Registers CTI Port/RP without Early Offer Support

*** App1 and App2 are running on Different Client Machines.

Action	TSP Message to application data
App1 and App2: Both Connecting to same CTI Manager lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
App1 and App2: lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
App1 and App2: LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
App1: Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
App1: Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to Register A	Line_Reply with Success LineInserviceEvent reports to Application.
App2: Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to Register A	Line_Devspecific fails with Error LINEERR_REGISTER_GETPORT_SUPPORT_MISMATCH
Variant: App1 and App2 connecting to different Cti Managers App2: (After App1 has already registered CtiPort -A) Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to register CtiPort A	LineReply – success LINE_CLOSE for the CTI Port

End-To-End Call Trace

Direct Call Scenario: Variation 1

Application does a LineInitializ. Application opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call.

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Direct Call Scenario: Variation 2

A calls B and B answers the call. Application does a LineInitialize. Application opens all lines with new ExtVersion 0x000A0000.

Action	CTI events	Expected results
A calls B. B answers the call		

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B with new ExtVesrion 0x000A0000	ExistingCallEvent received for A ExistingCallEvent received for A	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Consult Transfer Scenario: Variation 1

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. B sets up transfer to C, C answers the call, and B completes the transfer. A is connected to C.

Action	CTI event	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI event	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
B SetupTransfer to C	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0

Action	CTI event	Expected results
LineGetCallInfo on B (Consultation call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallInfo on C (Consultation call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
C answers the call. B completes transfer.	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2
LineGetCallInfo on A (Call between A and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C2 (Consultation call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Consult Transfer Scenario: Variation 2

A calls B and B answers the call. B sets up transfer to C. Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. Application completes the transfer. A is connected to C.

Action	CTI events	Expected Results
A calls B and B answers the call. B setups transfer to C and C answers the call	LineInitialize LineOpen on A , LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	

Action	CTI events	Expected Results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	ExistingCallEvent received for A (Primary Call between A and B)	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
	ExistingCallEvent received for B (Primary Call between A and B)	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B
	ExistingCallEvent received for B (Consultation Call between B and C)	LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received
	ExistingCallEvent received for C (Consultation Call between B and C)	dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A (Primary Call between A and B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B (Primary Call between A and B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected Results
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallInfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
Applications completes Transfer	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Blind Transfer Scenario

Application does a LineInitialize.Application opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. B does lineBlindTransfer to C. A is connected to C.

Action	CTI event	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI event	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
B lineBlindTransfer to C	NewCallEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Redirect Scenario

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. Application redirects B to C; A is connected to C.

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B with new ExtVesrion 0x000A0000		
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
B redirects call to C.C answers the call	NewCallEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Shared Line Scenario

Application does a LineInitialize. Application opens all lines with new ExtVersion 0x000A0000. A calls B, B'. B answers the call.

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B, LineOpen on B' with new ExtVesrion 0x000A0000		
A calls B	NewCallEvent received for A NewCallEvent received for B NewCallEvent received for B'	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B' LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on B'		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Shared Line Scenario with Barge

Application does a LineInitialize.Application opens all lines with new ExtVersion 0x000A0000. A calls B, B'. B answers the call.

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B, LineOpen on B' with new ExtVesrion 0x000A0000		
A calls B, B'answers the call	NewCallEvent received for A NewCallEvent received for B NewCallEvent received for B'	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B' LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on B'		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected results
B’ barges in	NewCallEvent received for B NewCallEvent received for B’ CallGlobalCallHandleChangedEvent received for B	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2 For B’ LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B3

Action	CTI events	Expected results
	CallGlobalCallHandleChangedEvent received for B’	For B’ LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B3
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B3
LineGetCallInfo on B’		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B3

Call Park Scenario: Variation 1

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. Application initiates a CallPark on B. A is parked on parkedDn. C calls parkDn and C is connected to A

Service Parameter Preserve globalcallid For Parked Calls set to False

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
Application initiates linepark on B		

Action	CTI events	Expected results
C dials parkDn	NewCallEvent received for C CallGlobalCallHandleChangedEvent received for A	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1 For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A2
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A2
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Call Park Scenario: Variation 2

Application does a LineInitialize.Application opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. Application initiates a CallPark on B. A is parked on parkedDn. C calls parkDn and C is connected to A

Service Parameter Preserve globalcallid For Parked Calls set to True

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
Application initiates linepark on B		

Action	CTI events	Expected results
C dials parkDn	NewCallEvent received for C CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

3-Party Conference Call Scenario

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. B sets up conference to C, C answers the call, and B completes conference. A, B and C are in conference.

Note For all conference scenarios, conference call leg's Unique Call Reference ID is 0.

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected results
B setupConference to C	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallnfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
C answers the call. B completes conference	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected results
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Three-Party Conference Drop Down to Two-Party Call Scenario

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A calls B and B answers the call. B sets up conference with C, C answers the call, and B completes conference. A,B and C in conference. C drops from the conference.A connected to B.

Action	CTI events	Expected results
LineInitialize Call lineNegotiateVersion with LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected results
B setupConference to C	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallnfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
C answers the call. B completes conference	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2
C drops from conference		

Action	CTI events	Expected results
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Conference Chaining Scenario Using Join

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A, B and C are in Conference1. C, D and E are in another Conference2. Application sends CallJoinRequest to join both the conference calls.

E drops from the conference.

Action	CTI events	Expected results
A, B and C are in conference		For A Unique Call Reference ID = ID1 For B Unique Call Reference ID = ID2 For C Unique Call Reference ID = ID3
C, D and E are in conference		For C Unique Call Reference ID = ID4 For D Unique Call Reference ID = ID5 For E Unique Call Reference ID = ID6
Application Joins two confeences		No change in Unique Call Reference ID after join
E drops from Conference	CallGlobalCallHandleChanged received for D	For D LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0

Action	CTI events	Expected results
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference ID1
LineGetCallnfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference ID
LineGetCallnfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference ID3
LineGetCallInfo on D		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference ID7

Transfer Call Scenario via QSIP Without Path Replacement

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A in Cluster 1 calls B in Cluster 2, B answers the call, and B sets up transfer to C in Cluster 1. C answers the call and B completes the transfer. A connected to C.

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1

Action	CTI events	Expected results
B SetupTransfer to C	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallInfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
C answers the call.B completes transfer.		
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Transfer Call Scenario via QSIP with Path Replacement

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000. A in Cluster 1 calls B in Cluster 2, B answers the call and sets up transfer with C in Cluster 1. C answers the call amd B completes the transfer. A connected to C.

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
B SetupTransfer to C	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0

Action	CTI events	Expected results
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallInfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
C answers the call.B completes transfer	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Hunt List Scenario

LineGroup LG1,LG2 and LG3 configured with A,B and C. HuntList “Hunt_List” configured with Line Groups LG1,LG2 and LG3. Hunt Pilot “99999” configured with this HuntList.

Application does a LineInitialize. Application opens all lines with new ExtVersion 0x000A0000. D calls “99999”. Call is routed through A, B and C.

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C, LineOpen on D with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
D calls Hunt Pilot DN.Call is first offered to Phone A, followed by B and then C.	NewCallEvent received for D NewCallEvent received for A	For D LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
	NewCallEvent received for B NewCallEvent received for C	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on D		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference D1
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1

Action	CTI events	Expected results
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Call Pickup Scenario: Variation 1

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000.

B and C in same Call Pickup Group. Service Parameter, Auto Call Pickup Enabled, is set to False. A calls B and C presses the NewCall softkey followed by Call Pickup softkey. Call is redirected to C.

Same Behaviour for Group Pickup.

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0

Action	CTI events	Expected results
C presses NewCall softkey followed by Call Pickup softkey	NewCallEvent received for C NewCallEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Call Pickup Scenario: Variation 2

Application does a LineInitialize and opens all lines with new ExtVersion 0x000A0000.

B and C are in the same Call Pickup Group. Service Parameter Auto Call Pickup Enabled is set to True. A calls B, C presses NewCall softkey followed by Call Pickup softkey, and call is redirected to C.

Same Behaviour for Group Pickup.

Action	CTI events	Expected results
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000		

Action	CTI events	Expected results
A calls B	NewCallEvent received for A NewCallEvent received for B	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
C presses NewCall softkey followed by Call Pickup softkey	NewCallEvent received for C CallGlobalCallHandleChanged received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

EnergyWise Deep Sleep Mode Use Cases

Configuration

Line A on Cisco Unified IP Phones Series 9900, 7900, and 6900 phones connect to an EnergyWise Switch, LineNegotiate with supported extension 0x000B0000 or higher, in order to receive the reason code in dwparam2 of LINE_LINEDEVSTATE /PHONE_STATE EVENT. From the Device Administration page, Enable Power save and configure Power On and Power Off timers.

Verify EnergyWisePowerSavePlus Reason Code in LINEDEVSTATE Message

Verify EnergyWisePowerSavePlus Reason code in LINEDEVSTATE message, whenDevice unregisters when going into Deep sleep.

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	CiscoTSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
When Phone A goes to Deep Sleep mode and unregisters	Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0

Action	Expected result
When PowerOntime is reached, Cisco Unified IP Phones Series 7900 device registers back to CUCM.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0

Variance

For Cisco Unified IP Phones Series 9900 and 6900, press the Select Key to power up.

Verify EnergyWisePowerSavePlus Reason Code in PhoneState Suspend

Verify EnergyWisePowerSavePlus Reason code in PhoneState suspend, whenDevice unregisters when in Deep Sleep Mode.

Action	Expected result
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Phone A goes to Deep Sleep Mode and unregisters.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0

Action	Expected result
When PowerOntime is reached, Cisco Unified IP Phones Series 7900 device registers back to CUCM.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0

Variance

For Cisco Unified IP Phones Series 9900 and 6900, press the Select Key to power up.

Verify Reason EnergyWisePowerSavePlus Reason Code in LineDevstate/Phone State Message

Verify EnergyWisePowerSavePlus Reason code in LineDevstate/Phone State message, when unregisters after Power save idle time-out. Configure power save idle time-out = 20 mins(default = 1 hour).

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	

Action	Expected result
Phone goes to Deep Sleep Mode and unregisters	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0
For Cisco Unified IP Phones Series 9900 and 6900, press the Select Key to power up.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0, Cisco TSP Notifies DeviceInServiceEvent to application through Phone state Event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_RESUME param2 = x0, param3 = x0,

Action	Expected result
Power Save idle timer expires and device goes to Deep Sleep and unregisters	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0

Verify Call Manager Failure Reason Code in LineDevstate/Phone State Message

Verify CallManagerFailure Reason code in LineDevstate/Phone State message, when Device unregisters when Call Manager service is Restarted from serviceability page.

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	

Action	Expected result
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Restart Call Manager services from serviceability page.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_CallManagerFailure param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_CallManagerFailure param3 = x0

Action	Expected result
Call Manager Restart successful and device registers back	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0, Cisco TSP Notifies DeviceInServiceEvent to application through Phone state Event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_RESUME param2 = x0, param3 = x0

Verify DeviceUnregister Reason Code in LineDevstate/Phone State Event

Verify DeviceUnregister Reason code in LineDevstate/Phone State Event, when Device unregisters by manually unplugging the Ethernet cable from device.

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	

Action	Expected result
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Manually unplug the Ethernet cable from device.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_DeviceUnregistered param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_DeviceUnregistered param3 = x0

Action	Expected result
Device registers back after plugging back to the switch	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0, Cisco TSP Notifies DeviceInServiceEvent to application through Phone state Event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_RESUME param2 = x0, param3 = x0

Verify CTILinkFailure Reason Code in LineDevstate/Phone State Message

Verify CTILinkFailure Reason code in LineDevstate/Phone State message, when CTIManager services are stopped.

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	

Action	Expected result
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Stop CTI Manager services from serviceability page.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_CTILinkFailure param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_CTILinkFailure param3 = x0

Action	Expected result
Restart CTI Manager services	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0, Cisco TSP Notifies DeviceInServiceEvent to application through Phone state Event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_RESUME param2 = x0, param3 = x0

Extension Mobility Cross Cluster

Common Configuration

- User A has a device profile EM_Profile1 configured with Line1 in Cluster1 (home cluster)
- CiscoTSP uses CTIManager on Cluster1 (home cluster) in order to open provider

TAPI Application Does LineInitializeEx and EMCC User Logs Into a Device

Title	EMCC user logs in to a device
Description	Testing the scenario where TAPI Application does LineInitializeEx and EMCCUserLogin to a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is not in application control list
Expected Results	Step 2: Application receives LINE_CREATE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.

2. User A EM login to DeviceH on Cluster1.

TAPI Application Does LineInitializeEx and EMCCUser Logs Out of a Device

Title	EMCC user logs out of a device
Description	Testing the scenario where TAPI Application does LineInitializeEx and EMCCUserLogs out of a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is not in application control list
Expected Results	Step 2: Application receives LINE_REMOVE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM logout of a device DeviceH on Cluster1.

Application Does PhoneInitializeEx and EMCC User Logs In to a Device

Title	EMCC user logs in to a device
Description	Testing the scenario where TAPI Application does PhoneInitializeEx and EMCCUserLogin to a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is not in application control list
Expected Results	Step 2: Application receives PHONE_CREATE for Line1

1. Step1: Open the TAPI Application with User A and do PhoneInitializeEx.
2. Step2: User A EM login to DeviceH on Cluster1.

TAPI Application Does PhoneInitializeEx and EMCC User Logs Out of a Device

Title	EMCC user logs out of a device
Description	Testing the scenario where TAPI Application does PhoneInitializeEx and EMCCUserLogs out of a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is not in application control list
Expected Results	Step 2: Application receives PHONE _REMOVE for Line1

1. Step1: Open the TAPI Application with User A and do PhoneInitializeEx.
2. Step2: User A EM logout of a device DeviceH on Cluster1.

EMCC User Logs in to a Device From Cluster 2 (Visiting Cluster)

Title	EMCC user logs in to a device from cluster 2 (visiting cluster)
Description	Testing the scenario where EMCCUser Login to a Device from cluster 2 (visiting cluster)
Test Setup	EM_Profile1 is included in application control list.
Expected Results	Step 2: Application receives LINE_CREATE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A goes to the Cluster 2(visiting Cluster) and EM login to a device DeviceV.

EMCC User Logs Out of a Device From Cluster 2 (Visiting Cluster)

Title	EMCC user logs out of a device from cluster 2 (visiting cluster)
Description	Testing the scenario where EMCCUser LogOut of a Device from cluster 2 (visiting cluster)
Test Setup	EM_Profile1 is included in application control list.
Expected Results	Step 2: Application receives LINE_REMOVE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. After the Execution of the above usecase User A EM logout of a device DeviceV.

EMCC User Logs In to a Device with LineH Configured

Title	EMCC user logs in to a device with LineH configured
Description	Testing the scenario where EMCCUserLogin to a Device with LineH configured
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured
Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.

2. User A EM login to a device DeviceH on Cluster1.

EMCC User Logs Out of a Device with LineH Configured

Title	EMCC user logs out of a device
Description	Testing the scenario where EMCCUserLogs out of a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured
Expected Results	Step 2: • Application receives LINE_REMOVE for Line1 • Application receives LINE_CREATE for LineH

1. After the Execution of the above usecase User A EM logout of a device DeviceH on Cluster1.

EMCC User Logs In to a DeviceH Configured for Multiple Lines (LineH)

Title	EMCC user logs in to a DeviceH
Description	Testing the scenario where EMCCUser Login to a DeviceH which is configured for multiple lines
Test Setup	EM_Profile1 is included in application control list
Expected Results	Step 2: • Application receives 2 LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A goes to the Cluster 2(visiting Cluster) and EM login to a device DeviceH(A device with multiple lines (LineH)).

EMCC User Logs In to a Device with LineH Configured and Administrator Removes the Device From Application Control List

Title	EMCC user logs in to a device with LineH configured and the administrator removes the device from the Application Control list
Description	Testing the scenario where EMCCUserLogin to a device with LineH configured and administrator removes the device from the Application Control list
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured

Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1 Step3: • Application will not receive any events.

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM login to a device DeviceH on Cluster1.
3. Administrator removes the DeviceH from application control list.

EMCC User Logs In and Out of a Device with LineH Configured and Administrator Removes the Device From Application Control List

Title	EMCC user logs in and logs out of a device with LineH configured and Administrator removes the device from the Application Control List
Description	Testing the scenario where EMCCUserLogin to a Device with LineH configured and Administrator removes the device from the Application Control List
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured
Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1 Step3: • Application receives LINE_REMOVE for Line1 • Application receives LINE_CREATE for LineH Step4: • Application receives LINE_REMOVE for LineH

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM login to a device DeviceH on Cluster1.
3. User A EM logout of the device DeviceH on Cluster1.
4. Administrator removes the DeviceH from application control list.

EMCC User Logs in to a Device with LineH Configured and EM_Profile Not Included in Application Control List

Title	EMCC user logs in to a device with LineH configured and administrator removes the device from the Application Control list

Description	Testing the scenario where EMCCUserLogin to a device with LineH configured and administrator removes the device from the Application Control list
Test Setup	EM_Profile1 is not included in Application Control list DeviceH is included in Application Control list with LineH configured
Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1 Step3: • Application receives no events since EM_Profile1 is not in control list. Step4: • Application receives LINE_REMOVE for LineH

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM login to a device DeviceH on Cluster1.
3. Administrator removes the DeviceH from application control list.
4. User A EM logout of the device DeviceH on Cluster1.

EMCC User Logs In to a DeviceV and EM_Profile Is Removed by Administrator From Application Control List

Title	EMCC user logs in to a DeviceV and administrator removes the EM_Profile from the Application Control list
Description	Testing the scenario where EMCCUserLogin to a DeviceV and administrator removes the EM_Profile from Application Control list
Test Setup	EM_Profile1 is included in Application Control list.
Expected Results	Step 2: • Application receives LINE_CREATE for Line1 Step3: • Application receives LINE_REMOVE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM login to a DeviceV (Visiting Device).
3. Administrator removes the EM_Profile1 from application control list.

EMCC User Logs In to a Device Then Application Does Provider Open

Title	EMCC user logs in to a DeviceV

Description	Testing the scenario where EMCCUserLogin to a DeviceV(cluster2). Then the application does Provider Open
Test Setup	EM_Profile1 is included in Application Control list DeviceH is not in Application Control list
Expected Results	Step2: • DeviceV/Line1 will be included in TAPI device/line enumeration

1. User A EM login to DeviceV on Cluster2.
2. Open the TAPI Application with User A and do LineInitializeEx.

EMCC User Logs In to a DeviceV in Visiting Cluster and Administrator Adds the EM_Profile to Application Control List

Title	EMCC user logs in to a DeviceV in Visiting cluster and administrator adds the EM_Profile to the Application Control List
Description	Testing the scenario where EMCCUserLogin to a DeviceV in Visiting cluster and Administrator adds the EM_Profile to the Application Control list
Test Setup	EM_Profile1 is not included in Application Control list
Expected Results	Step 2: • Application will not receive any events as EM_Profile1 not in the Application Control list. Step3: • Application receives LINE_CREATE for Line1

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User B EM login to a DeviceV on Cluster2.
3. Administrator Adds the EM_Profile1 to the application control list.

Extension Mobility Memory Optimization Option

The following section describes common configuration and use cases for Early Offer Support.

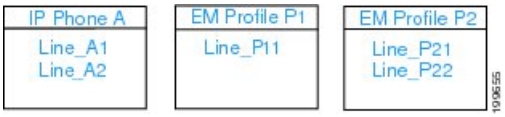
Common Configuration

The message flow in the following figure is described below:

- IP Phone_A is configured in DB with lines Line_A1 and LineA2
- User1 has a device profile EM_Profile1 configured with Line_P11

- User2 has a device profile EM_Profile2 configured with lines Line_P21 and Line_P22

Figure 1: EM Memory Optimization Scenario 1



The message flow in the following figure is described below:

- Application uses Line_N to receive other-device state notifications

Figure 2: EM Memory Optimization Scenario 2

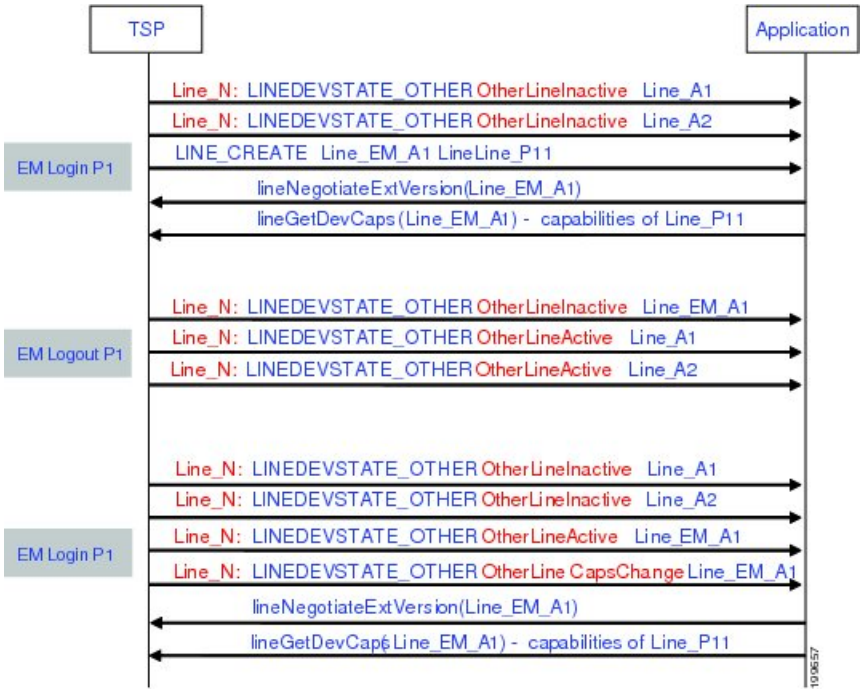
Use Cases

Use cases related to the EM Memory Optimization Option feature are mentioned below:

- Use Case 1
  1. Line_A1 and Line_A2 are not opened
  2. EM user with Profile_P1 logs in
  3. EM user with Profile_P1 logs out
  4. EM user with Profile_P1 logs in

The message flow in the following figure is described in steps 1 to 4.

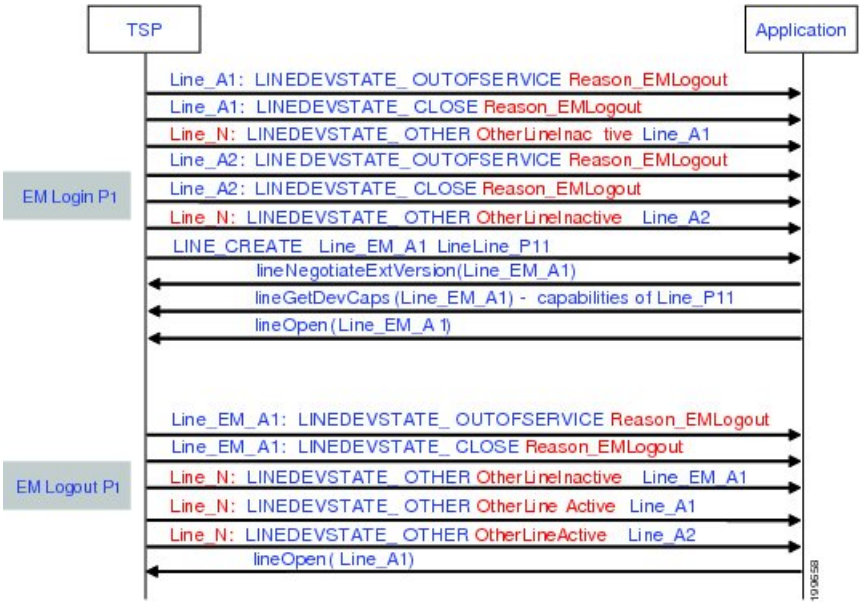
Figure 3: EM Memory Optimization Option Feature Use Case 1



- Use Case 2
  1. Line_A1 and Line_A2 has been opened
  2. EM user with Profile_P1 logs in
  3. Application opens Line_P11
  4. EM user with Profile_P1 logs out
  5. Application opens Line_A1

The message flow in the following figure is described in steps 1 to 5.

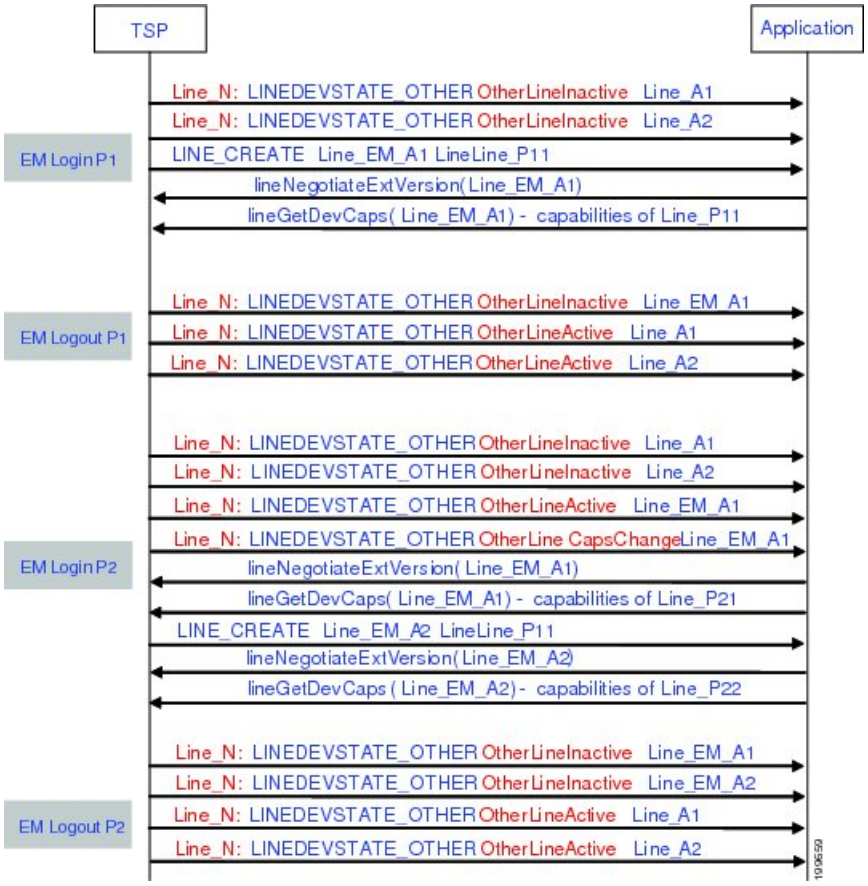
Figure 4: EM Memory Optimization Option Feature Use Case 2



- Use Case 3
  1. Line_A1 and Line_A2 are not opened
  2. EM user with Profile_P1 logs in
  3. EM user with Profile_P1 logs out
  4. EM user with Profile_P2 logs in
  5. EM user with Profile_P2 logs out

The message flow in the following figure is described in steps 1 to 5.

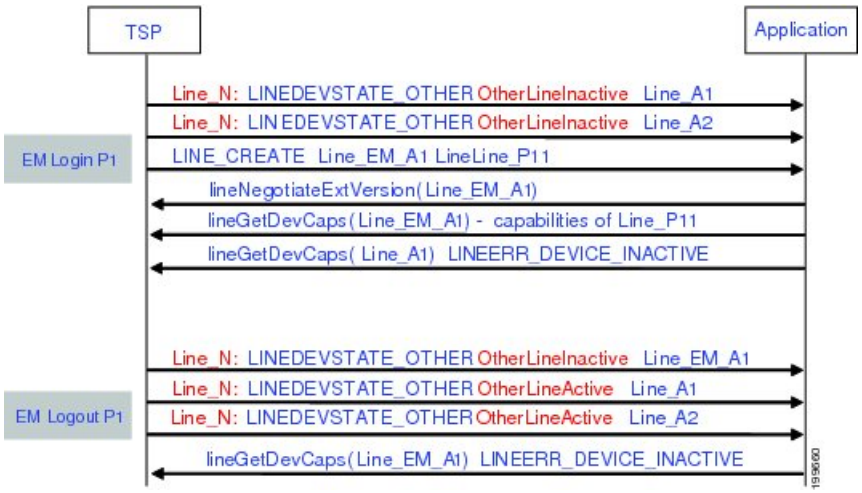
Figure 5: EM Memory Optimization Option Feature Use Case 3



- Use Case 4
  1. EM user with Profile_P1 logs in
  2. Operation request failed on inactive Line_A1
  3. EM user with Profile_P1 logs out
  4. Operation request failed on inactive Line_P11 with … error code …

The message flow in the following figure is described in steps 1 to 4.

Figure 6: EM Memory Optimization Option Feature Use Case 4



External Call Control

Basic Call Initiated From TAPI with 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 sends a lineMakeCall 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 Reject.

Party	TSP Message to App data
A initiates Call to B A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “” mod Calling = A / mod Called = “”

Party	TSP Message to App data
A receives CallStateChangeEvent (Disconnect)	A: LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED, LINEDISCONNECTMODE_REJECT) CallerID = A / CalledID = “” mod Calling = A / mod Called = “”

Basic Call Initiated From TAPI Using 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 sends a lineMakeCall 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 (modified calling and modified called in divert to routing directive, overrides the calling and called number transformation configured for translation pattern and the call is diverted to C)

Party	TSP Message to App data
A initiates call to B A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “” mod Calling = A / mod Called = “”
A receives CallStateChangeEvent (Proceeding)	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallerID = A / CalledID = B1 mod Calling = A1 / mod Called = B1

Party	TSP Message to App data
A receives CallStateChangeEvent (RingBack) C receives NewCallEvent	A: LINE_CALLSTATE (LINECALLSTATE_RINGBACK)/ LINE_CALLINFO CallerID = A / CalledID = B1 / RedirectingID = MB / RedirectionID = C mod Calling = MA / mod Called = B1 / mod Redirecting = MB / mod Redirection = C C: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED dwReason = LINECALLREASON_UNKNOWN extendCallReason = CtiReasonCallIntercept CallerID = A / CalledID = MB / RedirectingID = MB / RedirectionID = C mod Calling = MA / mod Called = MB / mod Redirecting = MB / mod Redirection = C
C answers A and C receives Connected Call state	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B1 / ConnectedID = C / RedirectingID = MB / RedirectionID = C mod Calling = MA / mod Called = B1 / mod Connected = C / mod Redirecting = MB / mod Redirection = C C: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED CallerID = A / CalledID = MB / ConnectedID = A / RedirectingID = MB / RedirectionID = C mod Calling = MA / mod Called = MB / mod Connected = MA / mod Redirecting = MB / mod Redirection = C

Basic Call Initiated From TAPI Using 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 sends a lineMakeCall 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 MB (modified calling and modified called in continue routing directive, overrides the calling & called number transformation configured for translation pattern)

Party	TSP Message to App Data
A initiates Call to B A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “” mod Calling = A / mod Called = “”
A receives CallStateChangeEvent (Proceeding)	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallerID = A / CalledID = B1 mod Calling = A1 / mod Called = B1

Party	TSP Message to App Data
A receives CallStateChangeEvent (RingBack) MB receives NewCallEvent	A: LINE_CALLSTATE (LINECALLSTATE_RINGBACK)/ LINE_CALLINFO CallerID = A / CalledID = B1 mod Calling = MA / mod Called = B1 MB: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED CallerID = A / CalledID = MB mod Calling = MA / mod Called = MB
MB answers A and MB receives Connected Call state	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B1 / ConnectedID = MB mod Calling = MA / mod Called = B1 / mod Connected = MB MB: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = MB / ConnectedID = A mod Calling = MA / mod Called = MB / mod Connected = MA

Conference Call Initiated From TAPI Using External Call Control on Translation Pattern and CEPM Returns Continue with Modified Calling and Called Parties in the Consult Call

Configuration

Phone A, B, C are in cluster devices.

C matches the translation pattern CXXX which has calling and called party transformation defined to transform B to A1 and C to C1 and External Call Control is also enabled.

Procedure

Application sends a lineMakeCall at A to call B. Application sends a lineSetupConference/lineAddToconference to B to consult conference the call to C.

Result

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

CEPM returns continue with ModifiedCalling = MB and ModifiedCalled = MC

Call will be extended to “MC” (modified calling and modified called in continue routing directive, overrides the calling & called number transformation configured for translation pattern)

After conference is complete, the correct number of CONFERENCE calls are see at all the participants.

Party	TSP Message to App Data
A and B receives Connected Call state	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B / ConnectedID = B mod Calling = A / mod Called = B / mod Connected = B B: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B / ConnectedID = A mod Calling = A / mod Called = B / mod Connected = A
B does a lineSetupConference / lineDial to call C. MC receives NewCallEvent	B: Call-1 LINE_CALLSTATE (LINECALLSTATE_ONHOLDPENDCONF) CallerID = A / CalledID = B / ConnectedID = A mod Calling = A / mod Called = B / mod Connected = A Call-2 LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallerID = B / CalledID = C1 mod Calling = MB / mod Called = C1 MC: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED) CallerID = B / CalledID = MC mod Calling = MB / mod Called = MC

Party	TSP Message to App Data
MC answers the call	B: Call-2 LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = B / CalledID = C1 / ConnectedID = MC mod Calling = MB / mod Called = C1 / mod Connected = MC MC: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = B / CalledID = MC / ConnectedID = B mod Calling = MB / mod Called = MC / mod Connected = MB

Party	TSP Message to App Data
B1 does a lineAddToConference	A: CONFERENCE CallerID = A / CalledID = B / ConnectedID = B mod Calling = A / mod Called = B / mod Connected = B CONNECTED CONFERENCE CallerID = A / CalledID = MC / ConnectedID = MC mod Calling = A / mod Called = MC / mod Connected = MC B: CONFERENCE CallerID = A / CalledID = B / ConnectedID = A mod Calling = A / mod Called = B / mod Connected = A CONNECTED CONFERENCE CallerID = B / CalledID = C1 / ConnectedID = MC mod Calling = B/ mod Called = C1 / mod Connected = MC MC: CONFERENCE CallerID = B / CalledID = MC / ConnectedID = B mod Calling = B / mod Called = MC / mod Connected = B CONNECTED CONFERENCE CallerID = MC / CalledID = A / ConnectedID = A mod Calling = MC / mod Called = A / mod Connected = A

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

Configuration

Phone A, C1, D are in cluster devices. B matches the translation pattern BXXX where External Call Control is enabled. Application sends a lineMakeCall at A to call B.

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

CUCM redirects the call to the hunt pilot C, and the chaperone member 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.

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.

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

Party	TSP Message to App Data
A initiates Call to B A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = "" mod Calling = A / mod Called = ""
A receives CallStateChangeEvent (Proceeding) webmail	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallerID = A / CalledID = B mod Calling = A / mod Called = B

Party	TSP Message to App Data
A receives CallStateChangeEvent (RingBack) C1 receives NewCallEvent	A: LINE_CALLSTATE (LINECALLSTATE_RINGBACK)/ LINE_CALLINFO CallerID = A / CalledID = B / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Redirecting = B / mod Redirection = C C1: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED CallerID = A / CalledID = B / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Redirecting = B / mod Redirection = C LINECALLINFO::DEVSPECIFIC would contain IsChaperoneCall = 0x1
C1 answers A and C1 receives Connected Call state	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B / ConnectedID = C / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Redirecting = B / mod Connected = B / mod Redirection = C C1: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = A / CalledID = B / ConnectedID = C / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Redirecting = B / mod Connected = B / mod Redirection = C
Application issues a lineRedirect on call at C1	Line_Reply is returned with an error code of LINEERR_OPERATION_FAIL_CHAPERONE_DEVICE

Party	TSP Message to App Data
C1 does a lineSetupConference / lineDial to call D. D receives NewCallEvent	C1: Call-1 LINE_CALLSTATE (LINECALLSTATE_ONHOLDPENDCONF) CallerID = A / CalledID = B / ConnectedID = A / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Connected = A / mod Redirecting = B / mod Redirection = C CONNECTED LINECALLINFO::DEVSPECIFIC would contain IsChaperoneCall = 0x1 Call-2 LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallerID = C1 / CalledID = D mod Calling = C1 / mod Called = D D: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_OFFERING/ LINECALLSTATE_ACCEPTED) CallerID = C1 / CalledID = D mod Calling = C1 / mod Called = D
D answers the call	C1: Call-2 LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = C1 / CalledID = D / ConnectedID = D mod Calling = C1 / mod Called = D / mod Connected = D D: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) CallerID = C1 / CalledID = D / ConnectedID = C1 mod Calling = C1 / mod Called = D / mod Connected = C1

Party	TSP Message to App Data
C1 does a lineAddToConference	

Party	TSP Message to App Data
	A: CONFERENCE CallerID = A / CalledID = B / ConnectedID = C / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Redirecting = B / mod Connected = C / mod Redirection = C CONNECTED CONFERENCE CallerID = A / CalledID = D / ConnectedID = D mod Calling = A / mod Called = D / mod Connected = D C1: CONFERENCE CallerID = A / CalledID = B / ConnectedID = A / RedirectingID = B / RedirectionID = C mod Calling = A / mod Called = B / mod Connected = A / mod Redirecting = B / mod Redirection = C CONNECTED LINECALLINFO::DEVSPECIFIC would contain IsChaperoneCall = 0x1 CONFERENCE CallerID = C / CalledID = D / ConnectedID = D mod Calling = C / mod Called = D / mod Connected = D D: CONFERENCE CallerID = C / CalledID = D / ConnectedID = C mod Calling = C / mod Called = D / mod Connected = C CONNECTED CONFERENCE

Party	TSP Message to App Data
	CallerID = D / CalledID = A / ConnectedID = A mod Calling = D / mod Called = A / mod Connected = A
Chaperone C1 starts recording to recording device R	C1: LINE_DEVSPECIFIC(SLDSMT_RECORDING_STARTED, 0, 0) LINE_DEVSPECIFIC(SLDSMT_LINECALLINFO_ DEVSPECIFICDATA, SLDST_CALL_ATTRIBUTE_INFO, 0) CallAttributeTye = ‘Recording’ C1’s CCMCallId Address = R’s DN, Partition = R’s Partition, DeviceName = R’s DeviceName

Forced Authorization and Client Matter Code Scenarios

Manual Call to a Destination That Requires an FAC

The following table describes the message sequences for the Forced Authorization and Client Matter Code scenario of Manual Call to a Destination that requires an FAC.

Preconditions

Party A is Idle. Party B requires an FAC.

The scenario remains similar if Party B requires a CMC instead of an FAC.

Table 51: Message Sequences for Manual Call to a Destination That Requires an FAC

Actions	CTI Message	TAPI messages	TAPI structures
Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A dials Party B	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = True, CMCRequired = False	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_FACREQUIRED	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A dials the FAC, and Party B accepts the call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change

Manual Call to a Destination That Requires Both FAC and CMC

The following table describes the message sequences for the Forced Authorization and Client Matter Code scenario of a manual call to a destination that requires both FAC and CMC.

Preconditions

Party A is Idle. Party B requires an FAC and a CMC.

Table 52: Message Sequences for Manual Call to a Destination That Requires Both FAC and CMC

Actions	CTI Message	TAPI messages	TAPI structures
Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A dials Party B	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = True, CMCRequired = True	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_FACREQUIRED, CZIPZIP_CMCREQUIRED	No change
Party A dials the FAC	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = False, CMCRequired = True	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_CMCREQUIRED	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A dials the CMC, and Party B accepts the call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change

lineMakeCall to a Destination That Requires an FAC

The following table describes the message sequences for the Forced Authorization and Client Matter Code scenario of lineMakeCall to a destination that requires an FAC.

Preconditions

Party A is Idle. Party B requires an FAC. Note that the scenario is similar if Party requires a CMC instead of an FAC.

Table 53: Message Sequences for lineMakeCall to a Destination That Requires an FAC

Actions	CTI Message	TAPI messages	TAPI structures
Party A does a lineMakeCall() to Party B	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = ORIGIN dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = REASON, CALLERID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = True, CMCRequired = False	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_FACREQUIRED	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A does a lineDial() with the FAC in the dial string and Party B accepts the call	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change

lineMakeCall to a Destination That Requires Both FAC and CMC

The following table describes the message sequences for the Forced Authorization and Client Matter Code scenario of lineMakeCall to a destination that requires both FAC and CMC. In this scenario, Party A is Idle and Party B requires both an FAC and a CMC.

Table 54: Message Sequences for lineMakeCall to a Destination That Requires Both FAC and CMC

Actions	CTI Message	TAPI messages	TAPI structures
Party A does a lineMakeCall() to Party B	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = ORIGIN dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = REASON, CALLERID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = True, CMCRequired = True	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_FACREQUIRED, CZIPZIP_CMCREQUIRED	No change

Actions	CTI Message	TAPI messages	TAPI structures
Party A does a lineDial() with the FAC in the dial string	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = False, CMCRequired = True	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_CMCREQUIRED	No change
Party A does a lineDial() with the CMC in the dial string and Party B accepts the call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change

Timeout Waiting for FAC or Invalid FAC

The following table describes the message sequences for the Forced Authorization and Client Matter Code scenario of timeout waiting for FAC or invalid FAC entered. Here, Party A is Idle and Party B requires an FAC.

The scenario remains similar if Party B required a CMC instead of a FAC.

Table 55: Message Sequences for Timeout Waiting for FAC or Invalid FAC

Actions	CTI Message	TAPI messages	TAPI structures
Party A does a lineMakeCall() to Party B	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = ORIGIN dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = REASON, CALLERID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	CallToneChangedEvent, CH = C1, Tone = ZipZip, Feature = FACCMC, FACRequired = True, CMCRequired = False	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_CALL_TONE_CHANGED dwParam2 = CTONE_ZIPZIP dwParam3 = CZIPZIP_FACREQUIRED	No change

Actions	CTI Message	TAPI messages	TAPI structures
T302 timer times out waiting for digits, or Party A does a lineDial() with an invalid FAC	CallStateChangedEvent, CH = C1, State = Disconnected, Cause = CtiNoRouteToDDestination, Reason = FACCMC, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DISCONNECTED dwParam2 = DISCONNECT MODE_FACCMC[1] dwParam3 = 0	No change
	CallStateChangedEvent, CH = C1, State = Idle, Cause = CtiCauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	No change

[1] dwParam2 get set to DISCONNECTMODE_FACCMC if the extension version on the line is set to at least 0x00050000. Otherwise, dwParam2 get set to DISCONNECTMODE_UNAVAIL.

Gateway Recording

Table 56: ClusterID and RecordType in LineGetDevCaps

Action	TSP Messages/Events
Application opens the provider.	
Application sends lineGetDevCaps on a line on the CTI Remote Device	LINEGETDEVCAPS::DEVSPECIFIC contains Cisco_LineDevCaps_Ext00080000::recordType = configured recording type Cisco_LineDevCaps_Ext000D0000::clusteID = cluster ID of the line

Setup:

A is external caller.

CTI RD has remote destination routed externally through a gateway that does not support recording

Table 57: External Call to a CTI Remote Device Using Ingress Gateway for Forking with Selective Recording

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	

Action	TSP Messages/Events
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A is external caller.

CTI RD has remote destination routed externally through a gateway that supports recording

Table 58: External Call to a CTI Remote Device Using Egress Gateway for Forking with Automatic Recording

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_Automatic (6) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A is external caller.

CTI RD has remote destination routed externally through a gateway that supports recording

Table 59: Initiate a Recording at CTIRD Follow by Hold and Resume the Call at the CTIRD

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_Automatic (6) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
CTI RD puts the call on hold	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event
CTI RD resumes the call	TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_Automatic (6) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A is external caller.

CTI RD has remote destination routed externally through a gateway that supports recording

Table 60: Initiate a Recording at CTIRD Follow by Hold and Resume the Call at the Internal Other Party

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A puts the call on hold	No events pass by TSP, recording continue
A resumes the call	No events pass by TSP, recording continue

Setup:

A, B are internal callers to the CTI RD

CTI RD has remote destination routed externally through a gateway that supports recording

Table 61: Initiate a Recording at CTIRD Follow by Internal Other Party Redirects the Call to an Internal 3rd Party

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	

Action	TSP Messages/Events
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A redirects the call to B	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event
B answers the call	TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A, B are external callers to the CTI RD through a SIP trunk

CTI RD has remote destination routed externally through a gateway that supports recording

Table 62: Initiate a Recording at CTIRD Follow by External Other Party Redirects the Call to an External 3rd Party

Action	TSP Messages/Events
Application opens the provider.	

Action	TSP Messages/Events
A calls the CTI RD, remote destination answers	
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A redirects the call to B	
B answers the call	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A, B are internal callers to the CTI RD

CTI RD has remote destination routed externally through a gateway that supports recording

Table 63: Initiate a Recording at CTIRD Follow by Internal Other Party Transfers the Call to an Internal 3rd Party

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A setup transfer to B	
B answers the call	
A completes the transfer to B	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A, B are external callers to the CTI RD through a SIP trunk

CTI RD has remote destination routed externally through a gateway that supports recording

Table 64: Initiate a Recording at CTIRD Follow by External Other Party Transfers the Call to an External 3rd Party

Action	TSP Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A setup transfer to B	
B answers the call	

Action	TSP Messages/Events
A completes the transfer to B	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A, B are internal callers to the CTI RD

CTI RD has remote destination routed externally through a gateway that supports recording

Table 65: Initiate a Recording at CTIRD Follow by Internal Other Party Conferences an Internal 3rd Party

Action	CTI Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	

Action	CTI Messages/Events
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked
A setup conference to B	
B answers the call	
A completes the conference to B	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Action	CTI Messages/Events
B drops from the conference	TSP sends a LineDevSpecific(SLDSMT_RECORDING_ENDED) event TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forked

Setup:

A, B are internal callers to the CTI RD

CTI RD has remote destination routed externally through a gateway that supports recording

Table 66: Initiate a Recording at CTIRD Follow by Restart Recording That Fails

Action	CTI Messages/Events
Application opens the provider.	
A calls the CTI RD, remote destination answers	

Action	CTI Messages/Events
Application issues a CCiscoLineDevSpecificStartCallRecordingwith m_InvocationType = RecordingInvocationType_UserControlledRecording (2)	TSP sends a LINE_REPLY TSP sends a LineDevSpecific(SLDSMT_RECORDING_STARTED) event LINEGETCALLINFO::DEVSPECIFIC **CallAttributeInfo::** PartyDN = Recorder's DN PartyPartition = Recorder's Partition DeviceName = Recorder's Device Name CallAttributeType = CallAttribute_Recorded_UserInitiatedFromApp (8) **RecordingAttributeInfo_ExtD0::** ForkingDeviceType = MediaForkingType_GW (2) ForkingDeviceName = trunk name to gateway GatewayCallProtocolReference = Cisco GUID ForkingClusterName = clusterID where media is forkedforkingClusterID = clusterID where media is forked
A setup transfer to B	
B answers the call	
A completes the transfer to B	There are no recording resource available so TSP sends a LineDevSpecific(SLDSMT_RECORDING_FAILED) event Application needs to restart the recording
B setup transfer to C	
C answers the call	
B completes the transfer to C	No restart of recording by CTI Remote Device.

Hunt List

Phones -A, B, C and X

Hunt Pilots: HP1

Member LG1, LG2, LG3

HP2.

Member LG11, LG12, LG13 are CTI port

Pickup Group1 : has LG1, lG2, LG3, X

Pickup Group2: has HP1, X

TSP app opens all lines, otherwise will be stated in use case.

Basic Hunt List Call

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1 HuntPilot = HP1
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG2 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
Variance : perform the test with all HuntList algorithm Top-Down algorithm Circular algorithm Longest Idle Time algorithm	

Hunt List Call Moved to Next Member

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, Called Name = HP1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Events, requests and responses
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
LG2 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Hunt List Calls FWNA and FWNA Is Not Configured on HuntPilot

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Events, requests and responses
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG2 to LG3	Call at LG2 goes IDLE At LG3: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call is aborted since LG3 does not answer the call.	At A: call will go IDLE LINEDISCONNECTMODE_NOANSWER? At LG3: call will go IDLE LINEDISCONNECTMODE_NOANSWER ?

Hunt List Call FWNA with FWNA to B

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Events, requests and responses
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG2 to LG3	Call at LG2 goes IDLE At LG3: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call is FWNA to B, and B answer	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connectedid = B At LG3: call will go IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A Redirecting = HP1 Redirection = B

Hunt List Call Dropped When Hunt List Is Busy and FWB Is Not Configured

Action	Events, requests and responses
Make LG1, LG2, LG3 busy App initiates call from A to HP1	At A: Call disconnected after it is initiated. LINEDISCONNECTMODE_BUSY

Hunt List Call Is Forwarded When Hunt List Is Busy and FWB Is Configured to B

Action	Events, requests and responses
Make LG1, LG2, LG3 busy App initiates call from A to HP1 and the call is forwarded to B	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, Called Name = HP1 HuntPilot = HP1 At B: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 Redirecting = HP1 Redirection = B

HuntList Call Redirected When in ACCEPT State

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
LG1 redirects call to B	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: Call goes IDLE At B: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 RedirectingID = HP1 RedirectionID = B

Hunt List Call Redirected When in Connected State

Table 67: Message Sequence for Hunt List Call Redirected When in Connected State

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 redirects call to B	At A : LINE_CALLSTATE -RINGBACK Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 RedirectingID = LG1 RedirectionID = B At LG1: Call goes IDLE At B: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 RedirectingID = LG1 RedirectionID = B

Hunt List Call Member Is CTI or RP Port

Action	Events, requests and responses
Same as 8.1, but with CTI port	Similar expectation

Hunt List Call Moved to Different Line Group Members and Answered by CTI Port

Table 68: Message Sequence for Hunt List Call Moved to Different Line Group Members and Answered by CTI Port

Action	Events, requests and responses
Same as 8.2, but with CTI port	Similar expectation

Hunt List Call Is Redirected to Another Hunt List

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
A redirects the call to HP2 and call offered to LG11	At A: Call goes IDLE At LG1: LINE_CALLSTATE -RINGBACK Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A At LG11: LINE_CALLSTATE -ACCEPTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 RedirectionID = HP2 RedirectingID = A

Action	Events, requests and responses
LG11 answers the call	At LG1: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectingID = A RedirectionID = HP2 At LG11: LINE_CALLSTATE -OFFERING Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A

Hunt List Call Is Consult Transferred to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup transfer to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = LG1

Action	Events, requests and responses
LG1 completes transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = LG1 At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = LG1

Hunt List Call Direct Transferred to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1
LG1 calls to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 performs Direct Transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = LG1 At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = LG1

Hunt List Call Is Conferenced to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Hunt List Call Is Joined to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 calls B, B answers the call	At LG1 Call-1: ONHOLD Caller = A Called = HP1 HuntPilot = HP1 Connected = A Call-2: CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 performs Join	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Hunt List Call Is Conferenced to Another Hunt List After LG11 Answers

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP2 ->LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2

Action	Events, requests and responses
	At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1 HP name = -empty CONFERECED Caller = LG11 Called = A Connected = A

Hunt List Call Conferenced to the Same Hunt List and Completes Conference Before Hunt List Agent Answers

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to HG1, where alerting on LG2,	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A RINGBACK Caller = LG1 Called = HP1 HuntPilot = HP1 At LG2: LINE_CALLSTATE -ACCEPTED Caller = LG1 Called = HP2 HuntPilot = HP2

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = HP1 HuntPilot = HP1 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = HP1 HuntPilot = HP1

Action	Events, requests and responses
	At LG2: ACCEPTED CONFERECED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HG1 CONFERECED Caller = LG2 Called = A Connected = A

Action	Events, requests and responses
LG2 answers the call	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 Called Name = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 ConnectedName = LG2 HuntPilot = HP1 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 Called = A Connected = A

Action	Events, requests and responses
	At LG2: CONNECTED CONFERECED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HG1 CONFERECED Caller = LG11

Hunt List Basic Call with SharedLine

LG1’ is sharedline with LG1

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A At LG1': LINE_CALLSTATE -CONNECTED INACTIVE Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Hunt List Basic Call with DND-R Configured on LG1

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG2 since LG1 has DND enabled.. Then LG2 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Hunt List Call Put in Conference via Join Operation

Action	Events, requests and responses
B calls A, A answer	At A: Call-1 LINE_CALLSTATE -CONNECTED Caller = B Called = A Connected = B At G: LINE_CALLSTATE -CONNECTED Caller = B Called = A Connected = A

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: Call-1 is on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
Application initiates JOIN calls on A with final call as call-1	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = B Called = A Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = B Called = A Connected = A CONFERECED Caller = B Called = LG1 HuntPilot = HP1

Action	Events, requests and responses
	Connected = LG1 HuntPilot = HP1

Hunt List Call Is Picked Up From Pickup Group -G-Pickup Auto Pick Pp Is Enabled

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
Line X got notification of the call	Got call pickup notification of call offering at LG1
Line X does group pick from LG1	At A: LINE_CALLSTATE -CONNECTED Caller = X Called = HP1, HuntPilot = HP1 ConnectedID = X At X: LINE_CALLSTATE -PROCEEDING Caller = X Called = PickGroup# LINE_CALLSTATE -CONNECTED Caller = X Called = PickGroup#, ConnectedID = A

Hunt List Call Is Picked Up From Pickup Group When LG1 Is in Pickup Group 1 -Auto Pickup Disabled

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
Line X got notification of the call	Got call pickup notification of call offering at LG1
Line X does group pick from LG1	Original pickup call goes IDLE
X got server call about the pickup call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1, HuntPilot = HP1 ConnectedID = X At X: new call offered at X from server, and answer LINE_CALLSTATE -CONNECTED Caller = A Called = X ConnectedID = A

Hunt List Call Is Picked Up From Pickup Group When HP2 Is in Pickup Group 2 -Auto Pick Up Enabled

Action	Events, requests and responses
App initiates call from A to HP2 and the call is offered at LG11	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP2, HuntPilot = HP2 At LG11: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP2, HuntPilot = HP2
Line X got notification of the call	Got call pickup notification of call offering at HP2
Line X does group pick from HP2	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP2, HuntPilot = HP2 ConnectedID = X At X: LINE_CALLSTATE -CONNECTED Caller = X Called = PickGroup#, ConnectedID = A

Conferenced Hunt List Call Becomes Two-Party Call

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 Called Name = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2

Action	Events, requests and responses
	At LG11: CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 CONFERECED Caller = LG11 Called = A Connected = A
LG11 drops call	At A: Conf Parent call goes IDLE CONFERENCED call to LG11 goes IDLE CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: Conf Parent call goes IDLE CONFERENCED call to LG11 goes IDLE CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A At LG11: Calls go IDLE

Hunt List Broadcast Scenario (Broadcast Option Is Configured on HP1)

Action	Events, requests and responses
App initiates call from A to HP1, and call is offered at LG1, LG2 and LG3	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 At LG2: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 At LG3: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1

Note HP Broadcast is not supported when interacting with Call PickUp feature.

Hunt List Call Is Involved in c-Barge Conference

LG1’ is sharedline with LG1

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A At LG1': LINE_CALLSTATE -CONNECTED INACTIVE Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = B Connected = B At LG1': LINE_CALLSTATE -CONNECTED INACTIVE Caller = A Called = HP1 HuntPilot = HP1 Connected = A LINE_CALLSTATE -CONNECTED INACTIVE CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Called Name = B Connected = B Called Name = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
	At LG1': CONNECTED INACTIVE CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Action	Events, requests and responses
LG1' cBarges in	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B CONFERENCED Caller = A Called = LG1' Connected = LG1' At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1 Called = B Connected = B CONFERENCED Caller = LG1 Called = LG1' Connected = LG1'

Action	Events, requests and responses
	CONNECTED INACTIVE CONFERECED Caller = LG1’ Called = LG1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1’ Called = B Connected = B CONFERENCED Caller = LG1’ Called = A Connected = A At LG1’: CONNECTED CONFERECED Caller = LG1’ Called = LG1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1’ Called = B Connected = B CONFERENCED Caller = LG1’ Called = A Connected = A CONNECTED INACTIVE

Action	Events, requests and responses
	CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B CONFERENCED Caller = LG1 Called = LG1’ Connected = LG1’ At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A CONFERENCED Caller = B Called = LG1’ Connected = LG1’

Hunt List Feature Interact with Four-Party Conference

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HG2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2

Action	Events, requests and responses
	CONFERECED Caller = LG11 Called = A Connected = A
LG1 setup conference to X, X answers the call	At LG1: ONHOLDPENDINGCONFERENCE CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 CONNECTED Caller = LG1 Called = X Connected = X At X: CONNECTED Caller = LG1 Called = X Connected = LG1

Action	Events, requests and responses
LG1 completes conference	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 CONFERENCED Caller = A Called = X Connected = X At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 CONFERENCED Caller = LG1 Called = X

Action	Events, requests and responses
	Connected = X
	At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG1 CONFERECED Caller = LG11 Called = A Connected = A CONFERENCED Caller = LG11 Called = X Connected = X

Hunt Pilot Connected Number Feature

HP1 and HP2 are 2 Huntpilots with configuration "Display Line Group Member DN as Connected Party" set.

HP1: LG1, LG2, LG3(LineGroup/MemberDNs

HP2: LG4, LG5, LG6(LineGroups/MemberDNs

Table 69: Basic Hunt List Call

Action	Expected events
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1 HuntPilot = HP1

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = A ModifiedRedirectingID = ModifiedRedirectionID =

Table 70: Hunt List Call Moved to Next Member

Action	Expected events
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Expected events
LG2 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG2 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = A
Variance : perform the test with all HuntList algorithm Top-Down algorithm Circular algorithm Longest Idle Time algorithm	

Table 71: Hunt List Call Is Redirected When It Is in Connected State

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN:ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN :ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID =

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN:ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN :ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID =

Action	Expected events
LG1 redirects call to B	At A : LINE_CALLSTATE -RINGBACK Caller = A Called = HP1 HuntPilot = HP1 Connected = RedirectingID = HP1 RedirectionID = B CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = ModifiedRedirectingID = [LG1] ModifiedRedirectionID = B At LG1: Call goes IDLE At B: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 RedirectingID = HP1 RedirectionID = B CPN: ModifiedCalling = A ModifiedCalled = [LG1] Modifiedconnected = ModifiedRedirectingID = LG1 ModifiedRedirectionID = B

Table 72: Hunt List Call -member Is CTI / RP Port

Action	Expected events
Same as ,Table 69: Basic Hunt List Call, on page 384 but with CTI port	Similar expectation as of Basic Hunt Call.

Table 73: Hunt List Call Moved to Different Line Group Members and Answered by CTI Port

Action	Expected events
Same as ,Table 70: Hunt List Call Moved to Next Member, on page 386 but with CTI port	Similar expectation as of Hunt List call moved to next member.

Table 74: Hunt List Call Is Redirected to Another Hunt List

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Expected events
A redirects the call to HP2 and call offered to LG11	At A: Call goes IDLE At LG1: LINE_CALLSTATE -RINGBACK Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A CPN: ModifiedCalling = LG1 ModifiedCalled = HP1 Modifiedconnected = ModifiedRedirectingID = A ModifiedRedirectionID = HP2 At LG11: LINE_CALLSTATE -ACCEPTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 RedirectionID = HP2 RedirectingID = A CPN:ModifiedCalling = LG1 ModifiedCalled = HP2 Modifiedconnected = ModifiedRedirectingID = A ModifiedRedirectionID = HP2

Action	Expected events
LG11 answers the call	At LG1: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectingID = A RedirectionID = HP2 CPN: ModifiedCalling = LG1 ModifiedCalled = HP1 Modifiedconnected = LG11 ModifiedRedirectingID = A ModifiedRedirectionID = LG11 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A CPN: ModifiedCalling = LG1 ModifiedCalled = HP2 Modifiedconnected = LG1 ModifiedRedirectingID = A ModifiedRedirectionID = LG11

Table 75: Hunt List Call Is Consult Transferred to Another Line

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup transfer to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = LG1

Action	Expected events
LG1 completes transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = B ModifiedRedirectingID = LG1 ModifiedRedirectionID = B At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = HP1 CPN: ModifiedCalling = A ModifiedCalled = B Modifiedconnected = A ModifiedRedirectingID = LG1 ModifiedRedirectionID = B

Table 76: Hunt List Call Is Conferenced to Another Line

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Expected events
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Table 77: Hunt List Call Is Conferenced to Another Hunt List After LG11 Answers

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Expected events
LG1 setup conference to HP2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HP1

Action	Expected events
LG1 completes conference	

Action	Expected events
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED [A-LG1] Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED[LG1-LG11] Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: CONNECTED CONFERECED [LG11-LG1] Caller = LG1 Called = HP2 HuntPilot = HP2

Action	Expected events
	Connected = LG1
	CONFERECED [LG11-A] Caller = LG11 Called = A Connected = A

Caller Consult Transfer Call to Another Hunt List

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
A setup transfer to HP2, offered at LG11, LG11 anwser	At A Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = A Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: LINE_CALLSTATE -CONNECTED Caller = A Called = HP2 HuntPilot = HP2 Connected = A

Action	Events, requests and responses
A completes transfer	At LG1 : LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectionID = LG11 RedirectingID = A At A: both call goes IDLE At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = LG11 RedirectingID = A

Hunt Group Login Status

Use cases related to HuntGroup Login Status with extension feature are mentioned below:

Device A, B.

Application opens the device and the line and set the HuntGroup log in status from Login(1) to Logout(2)

Action	Expected Events
Application does LineInitialize LineOpen on A with new ExtVesrion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen with Extension Version as 0x00030000	PhoneInitialize Successful

Action	Expected Events
Application does phoneSetStatusMessages • dwPhoneStates = 0xffffff • dwButtonModes = 0xc • dwButtonStates = 0x1	Phone_State Success
The request to set the HuntLog Status is sent using CCiscoPhoneDevSpecificSetHuntGroupLoginStatus	PHONE_REPLY with Success
(CCiscoPhoneDevSpecificSetHuntGroupLoginStatus)... • PARAM: hPhone • PARAM: m_HuntGroupLoginStatus = 2 (Logout) • PARAM: returnCode	PHONE_STATE received with • PHONESTATE_CAPSCHANGE= 0x00400000 • PHONECAPS_DEVSPECIFIC_HUNTGROUP_LOGIN_STATUS = 1 • HuntGroupLoginStatus = 2

Application opens the device and the line and set the HuntGroup log in status from Login(1) to Login(1)

Action	Expected Events
Application does LineInitialize LineOpen on A with new ExtVesrion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen with Extension Version as 0x00030000	PhoneInitialize Successful
Application does phoneSetStatusMessages: • dwPhoneStates = 0xffffff • dwButtonModes = 0xc • dwButtonStates = 0x1	PHONE_STATE Success
The request to set the HuntLog Status is sent using CCiscoPhoneDevSpecificSetHuntGroupLoginStatus	PHONE_REPLY with Success
(CCiscoPhoneDevSpecificSetHuntGroupLoginStatus)... • PARAM: hPhone • PARAM: m_HuntGroupLoginStatus = 1 (Logout) • PARAM: returnCode	No event will be sent to the application.

Application opens the device and the line and get HuntGroupLogin status of the device using LineGetDevCaps

Action	Expected Events
Application does LineInitialize LineOpen on A with new ExtVesrion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen with Extension Version as 0x00030000	PhoneInitialize Successful
Application queries the capabilities by using LineGetDevCaps with the extension 0x000E0000 and gets back the HuntGroupLogin Status.	LineGetDevCaps Successful

Application opens the device and the line and set the HuntGroup Login Status field as any number not falling in the range of(0,1,2)

Action	Expected Events
Application does LineInitialize LineOpen on A with new ExtVesrion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen with Extension Version as 0x00030000	PhoneInitialize Successful
Application does phoneSetStatusMessages • dwPhoneStates = 0xffffff • dwButtonModes = 0xc • dwButtonStates = 0x1	PHONE_STATE Success
The request to set the HuntLog Status is sent using CCiscoPhoneDevSpecificSetHuntGroupLoginStatus	PHONE_REPLY with error
(CCiscoPhoneDevSpecificSetHuntGroupLoginStatus)... • PARAM: hPhone • PARAM: m_HuntGroupLoginStatus = 5 • PARAM: returnCode	LINEERR_INVALPARAM is returned to the application.

Application updates the HuntGroup Login Status on Unsupported Device.

Action	Expected Events
Application does LineInitialize LineOpen on A(Cti Route point) with new ExtVersion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen with Extension Version as 0x00030000	PhoneInitialize Successful

Action	Expected Events
Application does phoneSetStatusMessages • dwPhoneStates = 0xffffff • dwButtonModes = 0xc • dwButtonStates = 0x1	PHONE_STATE Success
The request to set the HuntLog Status is sent using CCiscoPhoneDevSpecificSetHuntGroupLoginStatus	PHONE_REPLY with error
(CCiscoPhoneDevSpecificSetHuntGroupLoginStatus)... • PARAM: hPhone • PARAM: m_HuntGroupLoginStatus = 1 • PARAM: returnCode	LINEERR_OPERATIONUNAVAIL is returned to the application.

Application calls to Hunt Pilot where the Hunt Member is logged into HuntGroup

Login.

Phones - A, B, C

Hunt Pilot - HP1

Member - LG1

LG1 has the members - Phone B and C.

B is Logged out of the huntGroup

Action	Expected Events
Application does LineInitialize LineOpen on A with new ExtVesrion 0x000E0000 LineOpen on B with new ExtVesrion 0x000E0000 LineOpen on C with new ExtVesrion 0x000E0000	LineInitialize successful
Application does PhoneInitialize and PhoneOpen on B with Extension Version as 0x00030000	PhoneInitialize Successful

Action	Expected Events
Application initiates call from A to HP(hunt pilot) and call is offered to LG1	At A: • LINE_CALLSTATE (DIALING) • Caller = A • Called = HP1 • Hunt Pilot= HP1 • LINE_CALLSTATE (PROCEEEDING)
B does not get the call as it is logged out of the HuntGroup	At C: • LINE_CALLSTATE (OFFERING) • Caller = A • Called = HP1
C starts to ring and accepts the call.	At C: • LINE_CALLSTATE (CONNECTED-ACTIVE) • Caller = A • Called = HP1 • Hunt Pilot= HP1 • Connected = A At A: • LINE_CALLSTATE (CONNECTED-ACTIVE) • Caller = A • Called = HP1 • Hunt Pilot= HP1 • Connected = LG1

Intercom

This configuration gets used for all the following use cases:

1. IPPhone A has two lines, line1 (1000) and line2 (5000). Line2 represents an intercom line. Speeddial to 5001 with label ìAssistant_1î gets configured.
2. IPPhone B has three lines, line1 (1001), line2 (5001), and Line3 (5002). Line2 and Line3 represent intercom lines. Speeddial to 5000 with label ìManager_1î gets configured on line2. Line 3 does not have Speeddial configured for it.

3. IPPhone C has two lines, line1 (1002) and line2 (5003). 5003 represents an intercom line that is configured with Speeddial to 5002 with label ìAssistant_5002î.
4. IPPhone D has one line (5004). 5004 represnts an intercom line.
5. CTIPort X has two lines, line1 (2000) and line2 (5555). Line2 represents an intercom line. Speedial to 5001 gets configured with label ìAssistant_1î.
6. Intercom lines (5000 to 5003) exists in same partition = Intercom_Group_1 and they remain reachable from each other. 5004 exists in Intercom_Group_2.
7. Application monitoring all lines on all devices.

Assumption: Application initialized and CTI provided the details on speeddial and lines with intercom line on all the devices. Behavior should act the same for phones that are running SCCP, and those that are running SIP.

Application Invoking Speeddial

Action	Events
LineOpen on 5000 & 5001 Initiate InterCom Call on 5000	For 5000 receive LINE_CALLSTATE cbInst = x0 param1 = x03000000 param2 = x1, ACTIVE param3 = x0, Receive StartTransmission event For 5001 receive LINE_CALLSTATE cbInst = x0 param1 = x03000000 param2 = x1, ACTIVE param3 = x0, Receive StartReception event Receive zipzip tone with reason as intercom

Agent Invokes Talkback

Action	Events
Continuing from the previous use case, 5001 initiates LineTalkBack from application on the InterCom call	For 5000 receive LINE_CALLSTATE device = x10218 param1 = x100, CONNECTED param2 = x1, ACTIVE param3 = x0, Receive StartReception event For 5001 receive LINE_CALLSTATE device = x101f6 cbInst = x0 param1 = x100, CONNECTED param2 = x1, ACTIVE param3 = x0, Receive StartTransmission event

Change the SpeedDial

Action	Events
Open line 5000 LineChangeSpeeddial request (speeddial to 5003, label = “Assistant_5003”)	The new speed dial and label is successfully set for the intercom line Receive LineSpeeddialChangeEvent from CTI Send LINE_DEVSPECIFIC to indicate that speeddial and label changed
Application issues LIneGetDevCaps to retrieve speeddial/label that is set on the line	TAPI returns configured speeddial/label that is configured on the line.

IPv6 Use Cases

The use cases related to IPv6 are provided below:

Register CTI Port with IPv4 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv4

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv4 only on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv4. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is able to register CTI Port with IPv4 address.

Register CTI Port with IPv6 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv6

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv6 only on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv6. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is not able to register CTI Port. TSP returns error LINEERR_OPERATIONUNAVAIL

Register CTI Port with IPv6 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv4_v6

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv4_v6 on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv6. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is not able to register CTI Port. TSP returns error LINEERR_OPERATIONUNAVAIL

IPv6 Phone A Calls IPv6 Phone B

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Open two lines A and B 3. Phone A which is IPv6 calls Phone B which is IPv6 4. Events at Phone B 1. While Media is established: • Events on phone A • Event on phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: Blank FarEndIPAddressIpv6: IPv6 address of A Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of B. ReceptionRTPDestinationAddress = IPv6 address of A. Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of A. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv4_v6 Phone Calls IPv6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Open two lines A and B 3. Phone A which is IPv4_v6 calls Phone B which is IPv6 4. Events at Phone B 1. While Media is established: • Events on phone A • Event on phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: IPv4 address of A FarEndIPAddressIpv6: IPv6 address of A Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of B. ReceptionRTPDestinationAddress = IPv6 address of A. Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of A. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv4 Phone Calls IPv6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Open two lines A and B 3. Phone A which is IPv4 calls Phone B which is IPv6 4. Events at Phone B 1. While Media is established: • Events on phone A • Event on phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: IPv4 address of A FarEndIPAddressIpv6: Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv4 address of MTP Resource. ReceptionRTPDestinationAddress = IPv4 address of A. Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv6 Phone Calls IPv4 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Open two lines A and B 3. Phone A which is IPv6 only calls Phone B which is IPv4 4. Events at Phone B 1. While Media is established: • Events on phone A • Event on phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: FarEndIPAddressIpv6: IPv6 address of A Do a GetLinecallInfo, LineCallInfo will contain the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of A. Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv4 address of MTP Resource. ReceptionRTPDestinationAddress = IPv4 address of B.

IPv6 Phone Calls IPv4_v6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Phone A which is IPv6 only calls Phone B which is IPv4_v6 only. 3. Open lines A and B 4. Events at Phone B 1. While Media is established: • Events on phone A • Event on phone B	Existing Call, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: FarEndIPAddressIpv6: IPv6 address of A Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of A. Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of Phone A. ReceptionRTPDestinationAddress = IPv6 address of B.

Common Device Configuration Device Mode Changes From IPv4_v6 to IPv4

Steps	Expected result
User changes the device configuration on common device configuration from IPv4_v6 to IPv4 only	Application receives LineDevSpecific for the opened CTI Ports/RP in the device config indicating that Addressing mode has changed. All lines registered as IPv6 get a LINE_CLOSE Event. Application can then re-register these lines later.

Common Device Configuration Device Mode Changes From IPv4 to IPv6

Steps	Expected result
User changes the device configuration on common device configuration from IPv4 only to IPv6 only	Application receives LineDevSpecific for the opened CTI Ports/RP in the device config indicating that Addressing mode has changed. All lines registered as IPv4 get a LINE_CLOSE Event. Application can then re-register these lines later.

Join Across Lines

Setup

Line A on device A

Line B1 and B2 on device B

Line C on device C

Line D on device D

Line B1’ on device B1’, B1’ is a shared line with B1

Join Two Calls From Different Lines to B1

Action	Expected events
A ‡ B1 is HOLD	For A
C ‡ B2 is connected	LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1: LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2

Action	Expected events
	For B1': LINE_CALLSTATE param1 = x100, CONNECTED, INACTIVE Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)
	For B1'
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C

Join Three Calls From Different Lines to B1

Action	Expected events
A ‡ B1 is hold,	
C ‡ B2 is hold	
D ‡ B2 is connected	For A:
	LINE_CALLSTATE

Action	Expected events
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2:
	LINE_CALLSTATE for call-1
	param1 = x100, HOLD Caller = C, Called = B2 , Connected = C
	LINE_CALLSTATE for call-2
	param1 = x100, CONNECTED Caller = D, Called = B2 , Connected = D
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For D:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = D, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	CONFERENCED Caller = A Called = D, Connected = D
	For B1
	CONNECTED

Action	Expected events
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D
	For B2
	Call-1 and call-2 will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = C Called = A, Connected = A
	CONFERENCED Caller = C Called = D, Connected = D
	For D
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = D Called = A, Connected = A
	CONFERENCED Caller = D Called = C, Connected = C
	For B1’
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D

Join Calls From Different Lines to B1 with Conference

Action	Expected events
A,B1,C in conference where B1 is controller	For A:
D‡ B2 Connected	
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = A Called = C, Connected = C

Action	Expected events
	For B1:
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For B2:
	LINE_CALLSTATE for call-1
	param1 = x100, CONNECTED Caller = D, Called = B2 , Connected = D
	For C:
	CONNECTED
	CONFERENCED Caller = C, Called = A, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For D:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = D, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	CONFERENCED Caller = A Called = D, Connected = D
	For B1
	CONNECTED

Action	Expected events
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = C Called = A, Connected = A
	CONFERENCED Caller = C Called = D, Connected = D
	For D
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = D Called = A, Connected = A
	CONFERENCED Caller = D Called = C, Connected = C
	For B1’
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D

Join Two Calls From Different Lines to B1 While B1 Is Not Monitored by TAPI

Action	Expected events
A ‡ B1 is HOLD,	
C ‡ B2 is connected	For A:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1

Action	Expected events
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
User issues join request from phone with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)

Join Two Calls From Different Lines to B2

Action	Expected events
A ‡ B1 is HOLD,	
C ‡ B2 is connected	For A:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	LINE_CALLSTATE

Action	Expected events
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C ??
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)
	For B1'

Action	Expected events
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C

Action	Expected events
A ‡ B1 is HOLD,	For A:
B1 issues setup conference	
C ‡ B2 is connected	
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	Primary call
	LINE_CALLSTATE
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	Consult call
	DIALTONE
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A

Action	Expected events
Application issues lineDevSpecific(SLDST_JOIN) with the call on B2 as survival call	For A:
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B2
	CONFERENCED Caller = A Called = C, Connected = C
	For B1
	Both calls will go IDLE
	For B2
	CONNECTED
	CONFERENCED Caller = B1, Called = A, Connected = A
	CONFERENCED Caller = C Called = B1, Connected = C
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B2 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B2)
	For B1’
	Calls go IDLE

B1 Performs a Join Across Line Where B1 Is Already in a Conference Created by A

Action	Expected events
A, B1, C are in a conference created by A	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C

Action	Expected events
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For A:
	B2 calls D, D answers
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For B2:
	Connected -Caller = B2, Called = D, Connected = D
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Connected -Caller = B2, Called = D, Connected = B2
B1 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = D, Connected = D
	For B1:
	Conference – Caller = A, Called = B1, Connected = B1

Action	Expected events
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = D, Connected = D
	For B2:
	Call will go IDLE
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = D, Connected = D
	For D:
	Conference – Caller = B1, Called = D, Connected = B1
	Connected
	Conference – Caller = D, Called = A, Connected = A
	Conference – Caller = D, Called = C, Connected = C

B2 Performs a Join Across Line Where B1 Is Already in a Conference Created by A

Action	Expected events
A,B1,C are in a conference created by A	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	For C:

Action	Expected events
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
B2 calls D, D answers	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For B2:
	Connected -Caller = B2, Called = D, Connected = D
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For D:
	Connected -Caller = B2, Called = D, Connected = B2
B2 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B2
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = D, Connected = D
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected

Action	Expected events
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = D, Connected = D
	For B2:
	Call will go IDLE
	For C:
	Conference – Caller = B2, Called = C, Connected = B2
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = D, Connected = D
	For D:
	Conference – Caller = B2, Called = D, Connected = B2
	Connected
	Conference – Caller = D, Called = A, Connected = A
	Conference – Caller = D, Called = C, Connected = C

B1 Performs a Join Across Line Where B1 Is in One Conference and B2 Is in a Separate Conference

Action	Expected events
A,B1,C are in conference1	For A (GCID-1):
D, B2, E are in conference2	
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1 (GCID-1):
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For C (GCID-1):
	Conference – Caller = B1, Called = C, Connected = B1

Action	Expected events
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For D (GCID-2):
	Conference – Caller = D, Called = B2, Connected = B2
	Connected
	Conference – Caller = D, Called = E, Connected = E
	For B2 (GCID-2):
	Conference – Caller = D, Called = B2, Connected = D
	Connected
	Conference – Caller = B2, Called = E, Connected = E
	For E (GCID-2):
	Conference – Caller = B2, Called = E, Connected = B2
	Connected
	Conference – Caller = E, Called = D, Connected = D
B1 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = CFB-2, Connected = CFB-2
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = CFB-2, Connected = CFB-2
	For B2:
	Call will go IDLE
	For C:

Action	Expected events
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = CFB-2, Connected = CFB-2
	For D:
	Connected
	Conference – Caller = D, Called = E, Connected = E
	conference – Caller = D, Called = CFB-1, Connected = CFB-1
	For E:
	Connected
	Conference – Caller = E, Called = D, Connected = D
	Conference – Caller = E, Called = CFB-1, Connected = CFB-1

Logical Partitioning

Use cases related to Logical Partitioning feature are mentioned below:

Basic Call Scenario

Basic Call scenario ; Logical partitioning Enabled = true	
Description	Basic Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) on one Geolocation A calls B: LineMakeCall on A Dails B (DN) Variant 1: B Geo-Location was not Configured;B(PSTN);Policy Config : Interior to Interior Variant 2: B (PSTN) on another GeoLocation
Expected Results	Variant 1: Call will be successful; Reason: LP_IGNORE. Variant 2: A goes to Proceeding State and then On A there will be a DISCONNECTED call state will be sent to application with cause as LINEDISCONNECTMODE_UNKNOWN.

Redirect Scenario

Redirect scenario ; Logical partitioning Enabled = true	
Description	Redirect Call failure due to Logical partitioning Feature Policy.
Test Setup	Two Clusters (Cluster1 and Cluster2) configured with logical partition policy that will restrict the VOIP calls from Cluster1 to PSTN calls on Cluster2. (vice versa PSTN to VIOP) A on Cluster1 (VOIP) B on Cluster2 (VOIP) C on Cluster2 (PSTN) A calls B B redirects the call to C
Expected Results	Operation fails with error code LINEERR_OPERATION_FAIL_PARTITIONING_POLICY. Error code is processed on Cluster2
Variants	For Forward Operation same behaviour will be observed.

Transfer Call Scenario

Transfer Call scenario ; Logical partitioning Enabled = true	
Description	Transfer Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) in one GeoLocation (GeoLoc 1) B (VOIP) in another GeoLocation(GeoLoc 2) C (PSTN)in same GeoLocation as B (GeoLoc 2) A calls B SetUpTransfer on B. On Consult Call at B; Dials C. Complete Transfer on B.
Expected Results	Operation fails with error code "LINEERR_OPERATIONUNAVAIL".
Variants	For Operation Adhoc Conference same behaviour will be observed.

Join Scenario

Join scenario; Logical partitioning Enabled = true	
Description	Join failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) in one GeoLocation (GeoLoc 1) B (VOIP) in another GeoLocation(GeoLoc 2) C (VOIP)in same GeoLocation as B (GeoLoc 2) D (PSTN) in same GeoLocation as B (GeoLoc 2) B has Three Calls 1. B -> A 2. B -> C 3. B -> D Variant 1: Join on B with B -> A as Primary Call. Variant 2: Join on B with B -> D as Primary Call. Variant 3: Join on B with B -> C as Primary Call.
Expected Results	Variant 1: A, B and C will be in conference. Variant 2: B, C and D will be in conference. Variant 3:Either A or D will be in conference with B and C.

Shared Line Scenario

CallPickUp scenario ; Logical partitioning Enabled = true	
Description	CallPickUp Failure due to Logical partitioning Feature Policy.
Test Setup	A (PSTN) on one Geolocation -GeoLoc1 B (VOIP) on one Geolocation -GeoLoc1 C (VOIP) on one Geolocation -GeoLoc2 A Dails B B Parks the call C does LineUnPark
Expected Results	Call will be successful on A and A' call will not be present
Variants	Shared line features like barge, cbarge, hold & remote resume should be disabled for calls.

CallPark: Retrieve Scenario

CallPickUp scenario ; Logical partitioning Enabled = true	
Description	CallPickUp Failure due to Logical partitioning Feature Policy.
Test Setup	A (PSTN) on one Geolocation -GeoLoc1 B (VOIP) on one Geolocation -GeoLoc1 C (VOIP) on one Geolocation -GeoLoc2 A Dails B B Parks the call C does LineUnPark
Expected Results	CallUpark Will fail with error code "LINEERR_OPERATIONUNAVAIL".

Basic Call Scenario

Basic Call scenario ; Logical partitioning Enabled = true	
Description	Basic Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) on one Geolocation A calls B: LineMakeCall on A Dails B (DN) Variant 1: B Geo-Location was not Configured;B(PSTN);Policy Config: Interior to Interior Variant 2: B (PSTN) on another GeoLocation
Expected Results	Variant 1: Call will be successful; Reason: LP_IGNORE. Variant 2: A goes to Proceeding State and then On A there will be a DISCONNECTED call state will be sent to application with cause as LINEDISCONNECTMODE_UNKNOWN.

Manual Outbound Call

The following table describes the message sequences for Manual Outbound Call when party A is idle.

Action	CTI messages	TAPI messages	TAPI structures
1. Party A goes off-hook	NewCallEven CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change
2. Party A dials Party B	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change

Action	CTI messages	TAPI messages	TAPI structures
3. Party B accepts call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change
4. Party B answers call	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStartReceptionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartReception dwParam2 = IP Address dwParam3 = Port	No change
	CallStartTransmissionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartTransmission dwParam2 = IP Address dwParam3 = Port	No change

Note LINE_DEVSPECIFIC events are sent only if the application has requested them by using lineDevSpecific().

Monitoring and Recording

Monitoring a Call

A (agent) and B (customer) get connected. BIB on A gets set to on.

Action	CTI messages	TAPI messages	TAPI structures
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
C(supervisor) issues start monitoring req with A's permanentLineID as input	NewCallEvent, CH = C3, GCH = G2, Calling = C, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = ORIGIN dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = REASON, CALLERID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectingID = NP
A's BIB automatically answers	Party C		
	CallStateChangedEvent, CH = C3, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = C, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = A dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP
	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	MonitoringStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_MONITOR_STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	Party C		
	LineCallAttributeInfoEvent, CH = C3, Type = 2 (MonitorCall_Target), CI = C1, Address = A's DN, Partition = A's Partition, DeviceName = A's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = A dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: CallAttribute_SilentMonitorCall_Target, CI = C1, DN = A's DN, Partition = A's Partition, DeviceName = A's Name
	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	LineCallAttributeInfoEvent, CH = C1, Type = 1 (MonitorCall), CI = C3 Address = C's DN, Partition = C's Partition, DeviceName = C's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: CallAttribute_SilentMonitorCall, CI = C3 DN = C's DN, Partition = C's Partition, DeviceName = C's Name
C drops the call	Party C		
	CallStateChangedEvent, CH = C3, State = Idle, Cause = CauseNoError, Reason = Direct, Calling = C, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	
	Party A		
	MonitoringEndedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_MONITOR_ENDED dwParam2 = DisconnectMode_Normal dwParam3 = 0	

Automatic Recording

Recording type on A (agent phone) is configured as Automatic. D is configured as a Recorder Device.

Action	CTI messages	TAPI messages	TAPI structures
A recieves a call from B, and A answers the call Recording session gets established between the agent phone and the recorder	Party A		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = B, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	RecordingStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_ STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP

Action	CTI messages	TAPI messages	TAPI structures
	LineCallAttributeInfoEvent CH = C1, Type = 3 (Automatic Recording), Address = D's DN, Partition = D's Partition, DeviceName = D's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: App Controlled Recording, DN = D's DN, Partition = D's Partition, DeviceName = D's Name

Application-Controlled Recording

A (C1) and B (C2) connect. Recording Type on A gets configured as 'Application Based'. D gets configured as a Recorder Device.

Action	CTI messages	TAPI messages	TAPI structures
A issues start recording request Recording session gets established between the agent phone and the recorder	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	RecordingStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_ STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	LineCallAttributeInfoEvent CH = C1, Type = 4 (App Controlled Recording), Address = D's DN, Partition = D's Partition, DeviceName = D's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_ DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_ INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: App Controlled Recording, DN = D's DN, Partition = D's Partition, DeviceName = D's Name

Action	CTI messages	TAPI messages	TAPI structures
A issues stop monitoring request	RecordingEndedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_ENDED dwParam2 = DisconnectMode_Normal dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP

NuRD (Number Matching for Remote Destination) Support

Park Monitoring

Use cases related to Park Monitoring feature are mentioned below:

Park Monitoring Feature Disabled

Setup:

The Park Monitoring message flag is disabled by default.

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001)

All lines are monitered by TSP

Action	Expected events
1. A(3000) calls B(3001) 2. B(3001) receives the call and parks the call	Application will not be notified about the New Parked call through LINE_NEWCALL event as the park Monitoring flag is disabled.

Park Monitoring Feature Enabled

Setup:

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001),C(3002)

All lines are monitered by TSP

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001).	Park Status Event on B: At Step 3: Application will be notified about the New Parked call through LINE_NEWCALL event At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. Application does a LineGetCallInfo.
1. A(3000) calls B(3001) **2.** B(3001) receives the call and parks the call at 5555	LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 2 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 2: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call at 5555 **4.** The Park Monitoring Reversion Timer expires while the call is still parked.	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 3 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 3: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No Retrieve destination configured on B(3001) as C(3002) **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked. **1.** The Park Monitoring Forward No Retrieve timer expires and now the call is forwarded to the Park Monitoring Forward No Retrieve Destination C(3002).	Park Status Event on B: At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded Application will receive the LINE_CALLSTATE event with callstate IDLE. The reason code CtiReasonforwardedNoRetrieve will be updated in the LINECALLINFO::dwDevSpecificData.ExtendedCallInfo. dwExtendedCallReason = CtiReasonforwardedNoRetrieve.
	Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 4: 1. The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). 2. A(3000) calls B(3001) 3. B(3001) receives the call and parks the call 4. A(3000) hangs up the call.	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Abandoned. Application will receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo.
	LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName TransactionIDID = Sub1. dwBearerMode: ParkStatus = 4 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 5: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call **4.** The Park Monitoring Reversion Timer Expires while the call is still parked. **5.** C(3002) retrieves the call	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Retrieved. Application will receive the LINE_CALLSTATE event with callstate IDLE.
	Application does a LineGetCallInfo. hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 5 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled: ParkedParty = 3000 dwCalledIDName: ParkedPartyPartition = P1.

Action	Expected events
Scenario 6: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination not configured. **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked **6.** The Park Monitoring Forward No Retrieve timer expires and the call is forwarded to the Parkers line.	Park Status Event on B At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded. Application will receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo.
	LineCallInfo will contain the following: hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled: ParkedParty = 3000 dwCalledIDName: ParkedPartyPartition = P1.

Action	Expected events
Scenario 7: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination configured as self(Parkers Line) **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked **6.** The Park Monitoring Reversion Timer Expires while the call is still parked	Park Status Event on B At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 7: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded. Application will receive the LINE_CALLSTATE event with callstate IDLE.
7. The Park Monitoring Forward No Retrieve timer expires and the call is forwarded to the Parkers line.	Application does a LineGetCallInfo. LineCallInfo will contain the following: hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Parked Call Exists

Setup:

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001).

B is not monitered by TSP.

Action	Expected events
Scenario 1: 1. The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). 2. A(3000) calls B(3001) 3. B(3001) receives the call and parks the call 4. Now the Line B(3001) is monitered by TSP	Park Status Event on B: At Step 4: Application will be notified about the Parked call through LINE_NEWCALL event.when ever cisco TSP recives the LINE_PARK_STATUS event for already parked call. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName TransactionIDID = Sub1. dwBearerMode: ParkStatus = 2 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Shared Line Scenario

Setup:

A(3000) ,D(3003) are Cisco Unified IP phones (future version) running SIP

B(3001) and B'(3001) are shared lines for Cisco Unified IP phones (future version) running SIP

C(3002) and C'(3002) are shared lines where C is a Cisco Unified IP phone (future version) running SIP and C' is a Cisco Unified IP Phone 7900 Series running SIP .

For the shared lines the events will be delivered to the phone which parks the call .Events will not be delivered to the other phone though the line is shared.

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) and B'(3001) starts ringing. B(3001) receives the call and parks the call **4.** Park Monitoring reversion timer expires while the call is still parked. **5.** D(3003) retrieves the call	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder.
	At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Retrieved Application will receive the LINE_CALLSTATE event with callstate IDLE.
	Application does a LineGetCallInfo. hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName :TransactionIDID = Sub1. dwBearerMode: ParkStatus = 5 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 2: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination configured as B(3001) **3.** A(3000) calls B(3001) **4.** B(3001) and B'(3001) starts ringing. B(3001)receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked. **6.** The Park Monitoring Forward No Retrieve timer expires and call is forwarded to B(3001).Both B(3001) and B'(3001) starts ringing as they are shared lines.	Park Status Event will be sent only to B not B'. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application receives the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application receives the LINE_CALLSTATE event with the Park Status = Forwarded. Application receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo. LineCallInfo contains the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.
Scenario 3: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls C(3002) **3.** C(3002) and C'(3002) starts ringing. C'(3002) receives the call and parks the call **4.** D(3003) retrieves the call	Park Status Event on C'. At Step 3: Application is notified about the New Parked call through LINE_NEWCALL event as the call is parked by the Normal TNP phone.

Park Monitoring Feature Disabled

Setup:

The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for line B(3001).

A(3000), D(3003) is a Cisco Unified IP phones (future version)

Application invokes the Line_open () API on provider to monitor ParkDN

.

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call **4.** The Park Monitoring Reversion Timer Expires while the call is still parked.	Park Status Event on B: At Step 3: Application receives the LINE_NEW_CALL event for PARKDN. At Step 3: Application receives the LINE_PARK_STATUS event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALL_STATE event with the Park Status = Reminder. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName :TransactionIDID = Sub1. dwBearerMode: ParkStatus = 3 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

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 78: 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 79: 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	
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 80: 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.

Action	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
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.getTerminal Connections() on device A.		Terminal.getTerminalConnections() on device A = null

Action	Events	Call Info
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 81: 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 82: 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]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[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
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 83: 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	

Actions	Events	Call Info
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", true) on device A	CiscoProvTerminal RemoteDestinationChangedEv	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
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
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", false) on device A.	CiscoProvTerminal RemoteDestinationChangedEv 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() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = false

Table 84: 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.	CiscoProvTerminal RemoteDestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[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.	CiscoProvTerminal RemoteDestinationChangedEv Both persistent call with GC1 and customer call with GC2 are not dropped/disconnected even though active rd = false.	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[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 85: 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 86: 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 87: 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 88: 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 89: 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 90: 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 91: 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 92: 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 93: 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.		
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 94: 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 95: 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 96: 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 97: 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.

Presentation Indication

Making a Call Through Translation Pattern

The following table describes the message sequences for the Presentation Indication scenario of making a call through translation pattern. In the Translation Pattern admin pages, both the callerID/Name and ConnectedID/Name get set to "Restricted".

Action	CTI messages	TAPI messages	TAPI structures
Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change
Party A dials Party B through Translation pattern	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
Party B accepts the call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPartyPI = Allowed, Called = B, CalledPartyPI = Restricted, OrigCalled = B, OrigCalledPI = restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCallerIDName = A's Name dwCalledID = B dwCalledIDName = B’s name dwConnectedID = NP dwConnectedIDName = NP dwRedirectionID = NP dwRedirectionIDName = NP dwRedirectionID = NP dwRedirectionIDName = NP

Action	CTI messages	TAPI messages	TAPI structures
Party B accepts the call (continued)	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPI = Allowed, Called = B, CalledPI = Restricted, OrigCalled = B, OrigCalledPI = Restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwRedirectionID = NP dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP
Party B answers the call	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPI = Allowed, Called = B, CalledPI = Restricted, OrigCalled = B, OrigCalledPI = Restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCallerIDName = A's Name dwCalledID = B dwCalledIDName = B’s Name dwConnectedID = A, dwConnectedIDName = A's Name, dwRedirectingID = NP dwRedirectingIDName = NP dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStartReceptionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartReception dwParam2 = IP Address dwParam3 = Port	No change
	CallStartTransmissionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC1 hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartTransmission dwParam2 = IP Address dwParam3 = Port	No change

Note LINE_DEVSPECIFIC events only get sent if the application requested them by using lineDevSpecific().

Blind Transfer Through Translation Pattern

The following table describes the message sequences for the Presentation Indication scenario of Blind Transfer through Translation Pattern. In this scenario, A calls via translation pattern B, B answers, and A and B are connected.

Action	CTI messages	TAPI messages	TAPI structures
Party B does a lineBlindTranfser() to blind transfer call from party A to party C via translation pattern	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CallingPartyPI = Restricted, CalledChanged = True, Called = C, CalledPartyPI = Restricted, OriginalCalled = NULL, OriginalCalledPI = Restricted, LR = NULL, Cause = BlindTransfer	LINE_CALLINFO, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = CONNECTEDID, REDIRECTINGID, REDIRECTIONID	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B’s name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B’s name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C2, State = Idle, Reason = Direct, Calling = A, CallingPartyPI = Restricted, Called = B, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = NULL	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B's name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party B does a lineBlindTranfser() to blind transfer call from party A to party C via translation pattern (continued)	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	NewCallEvent, CH = C3, origin = Internal_Inbound, Reason = BlindTransfer, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_APPNEWCALL hDevice = C dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = TRANSFER dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = NP dwCalledIDName = NP dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party C is offering	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangeEvent, CH = C1, State = Ringback, Reason = Direct, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B's name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party C is offering (continued)	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C3, State = Offering, Reason = BlindTransfer, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = OFFERING dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = NP dwCalledIDName = NP dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP

Redirect to Device

The following use cases are related to PSAP Callback Redirect to a device feature. For all use cases, there are four devices: device A, B, C and C'. Devices C and C' share a line.

Scenario 1: A calls B and B redirects the call to C, C' with redirectDeviceName as C.

Action	Expected Events
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C LineOpen on C ' with new ExtVersion 0x000D0000	

Action	Expected Events
A calls B	For A: • LINECALLSTATE CONNECTED • Caller = A, Called =B Connected B For B: • LINECALLSTATE CONNECTED • Caller = A, Called =B Connected A
Application sends CciscoLineDevSpecificRedirectEx on B to redirect call to C and C' with the redirectDeviceName as of C. • PARAM: hLine • PARAM: dwAddressID • PARAM: hCall • PARAM: FeaturePriority • PARAM: m_DestDirn • PARAM: m_SetOriginalCalledTo • PARAM: m_FAC • PARAM: m_CMC • PARAM: m_RedirectBitMask • PARAM: m_RedirectDeviceName = C • PARAM: m_ApplicationXMLDataSize • PARAM: m_ApplicationXMLData • PARAM: m_callingSearchSpace • PARAM: returnCode	For A: • LINECALLSTATE_RINGBACK For C: • LINECALLSTATE_OFFERING / LINECALLSTATE_ACCEPTED
C answers the call	For C: • LINE_CALLSTATE CONNECTED ACTIVE For C': • LINE_CALLSTATE CONNECTED INACTIVE

Scenario 2: A calls B and B redirects the call to C, C' with invalid device name.

Action	Expected Events
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C LineOpen on C ' with new ExtVesrion 0x000D0000	
A calls B	For A: • LINECALLSTATE CONNECTED • Caller = A, Called =B Connected B For B: • LINECALLSTATE CONNECTED • Caller = A, Called = B, Connected = A
Application sends CciscoLineDevSpecificRedirectEx on B to redirect call with invalid device name. (CciscoLineDevSpecificRedirectEx)... • PARAM: hLine • PARAM: dwAddressID • PARAM: hCall • PARAM: FeaturePriority • PARAM: m_DestDirn • PARAM: m_SetOriginalCalledTo • PARAM: m_FAC • PARAM: m_CMC • PARAM: m_RedirectBitMask • PARAM: m_RedirectDeviceName = invDevice • PARAM: m_ApplicationXMLDataSize • PARAM: m_ApplicationXMLData • PARAM: m_callingSearchSpace • PARAM: returnCode	Line_Reply with Error Code: "LINEERR_ INVALADDRESS"

Scenario 3: A calls B and B redirects the call to C,C' with redirectDeviceName as of C and with CallingSearchSpace with the value 2.

Action	Expected Events
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C LineOpen on C ' with new ExtVesrion 0x000D0000	
A calls B	For A: • LINECALLSTATE CONNECTED • Caller = A, Called =B Connected = B For B: • LINECALLSTATE CONNECTED • Caller = A, Called = B Connected = A
Application sends CciscoLineDevSpecificRedirectEx on B to redirect call to C and C' with the redirectDeviceName as of C. • PARAM: hLine • PARAM: dwAddressID • PARAM: hCall • PARAM: FeaturePriority • PARAM: m_DestDirn • PARAM: m_SetOriginalCalledTo • PARAM: m_FAC • PARAM: m_CMC • PARAM: m_RedirectBitMask • PARAM: m_RedirectDeviceName = C • PARAM: m_ApplicationXMLDataSize • PARAM: m_ApplicationXMLData • PARAM: m_callingSearchSpace=2 • PARAM: returnCode	For A: • LINECALLSTATE_RINGBACK For C: • LINECALLSTATE_OFFERING / LINECALLSTATE_ACCEPTED The CallingSearchSpace for device C will be set to the CSS of B (the party which is redirecting).

Action	Expected Events
C answers the call	For C: • LINE_CALLSTATE - CONNECTED ACTIVE For C' • LINE_CALLSTATE -CONNECTED INACTIVE

Redirect Set Original Called (TxToVM)

The following table describes the message sequences for Redirece Set Original Called (TxToVM) feature where A calls B, B answers, and A and B are connected.

Table 98: Message Sequences for Redirect Set Original Called (TxToVM)

Action	CTI messages	TAPI messages	TAPI structures
Party B does lineDevSpecific for REDIRECT_SET_ ORIG_CALLED with DestDN = C's VMP and SetOrigCalled = C	Party A		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CalledChanged = True, Called = C, OriginalCalled = NULL, LR = NULL, Cause = Redirect	LINE_CALLINFO, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = CONNECTEDID, REDIRECTINGID, REDIRECTIONID	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = NP dwRedirectionID = NP
	Party B		
	CallStateChangedEvent, CH = C2, State = Idle, reason = DIRECT, Calling = A, Called = B, OriginalCalled = B, LR = NULL	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = NULL dwRedirectionID = NULL
	Party C's VMP		
	NewCallEvent, CH = C3, origin = Internal_Inbound, reason = Redirect, Calling = A, Called = C, OriginalCalled = C, LR = B	TSPI: LINE_APPNEWCALL hDevice = C dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = REDIRECT dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C's VMP

Action	CTI messages	TAPI messages	TAPI structures
Party C is offering	Party A		
	CallStateChangeEvent, CH = C1, State = Ringback, Reason = Direct, Calling = A, Called = C, OriginalCalled = C, LR = B	TSPI: LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C's VMP
	Party C		
	CallStateChangedEvent, CH = C3, State = Offering, Reason = Redirect, Calling = A, Called = C, OriginalCalled = C, LR = B	TSPI: LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = OFFERING dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C

Refer and Replace Scenarios

In-Dialog Refer -Referrer in Cisco Unified Communications Manager Cluster

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer where referer is in Cisco Unified Communications Manager cluster.

Table 99: Message Sequences for In-Dialog Refer -Referrer in Cisco Unified Communications

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL		NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} LINECALLSTATE_OFFERING TAPI CallInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = “” dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
C answers the call, and Refer is successful	LINECALLSTATE_IDLE with extended REFER reason	CallPartyInfoChangedEvent @ B with {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = A dwRedirectionID = C dwConnectedID = C dwReason = DIRECT dwOrigin = LINECALL ORIGIN_INTERNAL	LINECALLSTATE_CONNECTED TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = B dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

In-Dialog Refer Where ReferToTarget Redirects the Call in Offering State

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer where ReferToTarget redirects the call in Offering state.

Table 100: Message Sequences for In-Dialog Refer Where ReferToTarget Redirects the Call In

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referee (B)	CallState/CallInfo @Refer-to-Target (C)
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	B gets CPIC with (calling = B, called = C, ocdpn = C, LRP = A, reason = REFER, call state = Ringback) TAPI CallInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = null dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} LINECALLSTATE_OFFERING TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = null dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL
C Redirects the call to D in offering state, and D answers	LINECALLSTATE_IDLE with extended reason = REFER (REFER considered as successful when D answers)	CallPartyInfoChangedEvent @ B with {calling = B, called = D, LRP = C, origCalled = C, reason = Redirect} Callstate = connected TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = C dwRedirectionID = D dwConnectedID = D dwReason = DIRECT dwOrigin = LINECALL ORIGIN_INTERNAL	IDLE with reason = Redirect TAPI LINECALLSTATE_IDLE D will get NewCallEvent with reason = Redirect call info same as B's call info. (calling = B, called = D, ocdpn = C, LRP = C, reason = redirect) Offering/accepted/connected

In-Dialog Refer Where Refer Fails or Refer to Target Is Busy

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer fails or refer to target is busy.

Table 101: Message Sequences for In-Dialog Refer Where Refer Fails or Refer to Target Is Busy

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B,) and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	No change	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
C is busy / C does not answer	A gets LINECALLSTATE_CONNECTED with extended reason = REFER (REFER considered as **failed)**	If B goes to ringback when call is offered to C (C does not answer finally) it should also receive Connected Call State and CPIC event TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	

Out-of-Dialog Refer

The following table describes the message sequences for the Refer and Replaces scenario of Out-of-Dialog Refer.

Table 102: Message Sequences for Out-of-Dialog Refer

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	There is no preexisting call between A and B.	There is no preexisting call between A and B.	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
A initiates REFER B to (C)		B should get NewCallEvent with call info as {calling = A, called = B, LRP = null, origCalled = B, reason = REFER} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_EXTERNAL	
B answers		Call state = connected (media does not flow between A and B when call goes to connected state) TAPI CallInfo (no change)	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Cisco Unified Communications Manager redirects the call to C		CallPartyInfoChangedEvent @ B with {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = A dwRedirectionID = C dwConnectedID = C dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_EXTERNAL	NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} This info is exactly same as though caller (A) performed REDIRECT operation (except the reason is different here). TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = B dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

Invite with Replace for Confirmed Dialog

The following table describes the message sequences for the Refer and Replaces scenario of invite with replace for confirmed dialog. Here, A, B, and C exist inside Cisco Unified Communications Manager. A confirmed dialog occurs between A and B. C initiates Invite to A with replace B's dialog ID.

Table 103: Message Sequences for Invite with Replace for Confirmed Dialog

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Confirmed dialog occurs between A and B	Call State = connected, Caller = A, Called = B, Connected = B, Reason = direct, gcid = GC1	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC1	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
C Invites A by replacing B's dialog			NewCall at C gcid = GC2, reason = REPLACEs, Call state = Dialing, Caller = C, Called = null, Reason = REPLACEs
Cisco Unified Communications Manager joins A and C in a call and disconnects call leg @ B	GCID Changed to GC2, Reason = REPLACEs CPIC Caller = C, Called = A, ocdpn = A, LRP = B Reason = REPLACEs Callstate = connected TAPI callinfo caller = C, called = B, connected = C, redirecting = B, redirection = A, reason = DIRECT with extended REPLACEs, callID = GC2	Call State = IDLE, extended reason = REPLACEs	CPIC changed Caller = C, Called = A, ocdpn = A, LRP = B, Reason = REPLACEs CallState = connected TAPI callinfo Caller = C, Called = A, Connected = A, Redirecting = B, Redirection = A, reason = UNKNOWN with extended REPLACEs, callID = GC2

Refer with Replace for All in Cluster

The following table describes the message sequences for the Refer and Replaces scenario of refer with replace for all in cluster. Here, a confirmed dialog exists between A and B and A and C. A initiates Refer to C with replace B's dialog ID.

Table 104: Message Sequences for Refer with Replace for All in Cluster

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referee (B)	CallState/CallInfo @Refer-to-Target (C)
Dialog between A and B and dialog between A and C	Call State = onhold, GC1, Caller = A, Called = C, Connected = C, Reason = direct CallState = connected, GC2, Caller = A, Called = B, Connected = B, Reason = direct	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC2	Call State = connected Caller = A, Called = C, Connected = A, Reason = direct, gcid = GC1
A completes Refer to C replacing A->B's dialog (B is referred to target)	From CTI (callState = IDLE with reason = TRANSFER) TAPI call state IDLE with Reason = DIRECT with extended reason TRANSFER	GCID changed from CTI reason = TRANSFER CPIC Changed from CTI Caller = B, Called = C, Origcalled = C, LRP = A, Reason = TRANSFER TAPI callinfo Caller = B, Called = B, Connected = C, Redirecting = A, Redirection = C, Reason = DIRECT with extended reason TRANSFER. CallId = GC1	CPIC Changed from CTI with Caller = B, Called = C, Origcalled = C, LRP = A, Reason = TRANSFER TAPI callinfo caller = B, called = C, connected = B, redirecting = A, redirection = C, reason = direct with extended TRANSFER. callId = GC1

Refer with Replace for All in Cluster Replace Dialog Belongs to Another Station

The following table describes the message sequences for the Refer and Replaces scenario of refer with replace for all in cluster, where replace dialog belongs to another station. In this scenario:

A is Referrer, D is Referee, and C is Refer-to-Target.

A confirmed dialog exists between A(d1) and B & C(d2) and D.

A initiates Refer to D on (d1) with Replaces (d2).

Table 105: Message Sequences for Refer with Replace for All in Cluster, Replace Dialog Belongs to Another Station

Actions	**CallState/CallInfo @Referrer (A)**	**CallState/CallInfo @B**	**CallState/CallInfo @Refer-to-Target (C)**	**CallState/CallInfo @Referree (D)**
Dialog between A and B and dialog between C and D	Call State = onhold, Caller = A, Called = B, Connected = B, Reason = direct, gcid = GC1	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC1	Call State = connected Caller = C, Called = D, Connected = D, Reason = direct, gcid = GC2	Call State = connected Caller = C, Called = D, Connected = C, Reason = direct, gcid = GC2
A initiates Refer to D on (d1) with Replaces (d2)	From CTI (callState = IDLE with reason = REFER) TAPI call state IDLE with reason = DIRECT with extended reason = REFER	CPIC Changed from CTI Caller = B, Called = C, Origcalled = D, LRP = C, Reason = REPLACEs TAPI callinfo Caller = B, Called = B, Connected = D, Redirecting = C, Redirection = D, Reason = DIRECT with extended REPLACEs, CallId = GC1	From CTI (callState = IDLE with reason = REPLACEs.) TAPI call state IDLE with reason = DIRECT with extended reason = REPLACEs	GCID changed from CTI to GC1 CPIC Changed from CTI with Caller = B (referee), Called = D, Origcalled = D, LRP = C, Reason = REPLACEs TAPI callinfo caller = B, called = D, connected = B, redirecting = C, redirection = D, reason = DIRECT with extended REPLACEs, callId = GC1

Secure Conferencing

Conference with All Parties as Secure

The conference bridge includes security profile. MOH is not configured. A, B, and C get registered as Encrypted.

Action	CTI messages	TAPI messages	TAPI structures
A calls B; B answers the call	Party A		
	CallStateChangedEvent, CH = C1, GCH = G1, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = A, CH = C1 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party B		
	CallStateChangedEvent, CH = C2, GCH = G1, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
B does lineSetUpConference	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
B calls C; C answers the call	Party B		
	CallStateChangedEvent, CH = C3, GCH = G2, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = B, CH = C3 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = C dwConnectedID = C dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C4, GCH = G2, Calling = B, Called = C, OrigCalled = C, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = C, CH = C4 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = C dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
B completes conf	Party B		
	CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted

Hold or Resume in Secure Conference

Conference bridge includes security profile. MOH gets configured. A, B, and C represent secure phones and exist in conference with overall call security status as secure.

Action	CTI messages	TAPI messages	TAPI structures
A does lineHold	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
	Party B		
	CtiCallSecurityStatusUpdate, LH = B, CH = C2, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
A does lineResume	Party A		
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = B, CH = C2, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party C		
	CtiCallSecurityStatusUpdate, LH = C, CH = C4, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted

Secure Monitoring and Recording

Silent Monitoring

Set up:

User is in “Allow Monitoring” Group

BIB on B is set to ON

A, A1 – Customer Phones

B, B1– Agent phones

C, C1 – Supervisor phones

All Lines are Opened with Ext Version – 0x000A0000

Action	Expected result
LineInitialize. Device A,B and C is Non-Secure LineOpen on A,B and C A calls B;B answers the Call C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on B LineGetCallInfo on C	Silent Monitored Call is created in Non-Secure Mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = CallAttributeInfo) will be fired to B and C CallReason = LINECALLREASON_DIRECT CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeType = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated CallReason = LINECALLREASON_DIRECT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info Extended Call Reason = "CtiReasonSilentMonitoring" CallAttributeType = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated
Varaint 1 : Monitor Customer, Agent and Supervisor Lines after Monitoring Session is Started. **Note** Start Monitoring Lines from Other Application or Close Agent and Supervisor and Reopen the same. LineGetCallInfo on B	CallReason = LINECALLREASON_UNKNOWN

Basic Silent Monitoring Scenario in Secure Mode

Action	Expected result
LineInitialize. Device A,B and C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on B LineGetCallInfo on C	Silent Monitored Call is created in Secure Mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C. SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Encrypted SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Silent Monitoring Scenario on Non-Secure Call in Secure Mode

Action	Expected result
LineInitialize. Device A is not Secure Device B and C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is non Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on B LineGetCallInfo on C Variant : A is Secure Call on A is Hold and Non-Secure MOH is Inserted	Monitoring Session will be started and the Media is setup in Secure Mode Events delivered will be same as use case 8.13.6.2. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C. SRTP info is not Available security Indicator = MEDIA_NOT_ENCRYPTED CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated SRTP info will be available security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated Same Events as above

Silent Monitoring Scenario on Non-Secure Call From Supervisor Which Is Secure

Action	Expected result
LineInitialize. Device A and B is not Secure Device C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is non Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on C	Call between B and C will be Non-Secure No SRTP Events will be fired CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info security Indicator = MEDIA_NOT_ENCRYPTED CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Silent Monitoring Scenario on Secure Call From Supervisor Which Is Non-Secure

Action	Expected result
LineInitialize. Device A and B is Secure Device C is Not Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	• New Call will be Fired on C. • Call on C will go to Disconnected State • Request fails with new Error Code LINEERR_SECURITY_CAPABILITIES_MISMATCH. **Note** Request fails as the Supervisor Security Capabilities doesn't meet or exceed the Security status of Agent (B)

Transfer of Monitored Call From Supervisor to Other Supervisor

Action	Expected result
LineInitialize. Device A,B and C is Secure Device C1 is not Secure LineOpen on A,B,C and C1 A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on C lineDevSpecifc(CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C	Call between B and C will be in Secure Mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C SRTP info will be available security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted LINE_REPLY (dwRequestId, 0) is returned CallSecurityStatus = Encrypted

Action	Expected result
C Transfers to C1 Variant : C1 is Secure LineGetCallInfo on B LineGetCallInfo on C1	Transfer is successful and Monitoring Session will be Terminated. Call on C1 will be Disconnected with new Cause Code. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE. Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C. Transfer is successful and Monitoring Session will not be disturbed. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C1 SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C1’s info. CallAttributeTye = ‘CallAttribute_SilentMonitorCall’ Address = C1’s DN, Partition = C1’s Partition Device Name = C1’s Device Name Transaction ID = XXXX Call Security Status = Encrypted SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of C will contain B’s info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B’s DN, Partition = B’s Partition Device Name = B’s Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Transfer of Call From One Customer to Other

Action	Expected result
LineInitialize. Device A,B and C is Secure Device A1 is not Secure LineOpen on A,B,C and A1 A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. A Transfers to A1 LineGetCallInfo on B LineGetCallInfo on C	Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C Call between B and C will be in Secure Mode Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C. Transfer is successful and Monitoring Session isn't disturbed. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = SLDST_SECURITY_STATUS_INFO) will be fired to B and C. SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C1's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated SRTP info will be available Security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Park on Supervisor

Action	Expected result
LineInitialize Device A,B and C is Secure Device C1 non secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input C parks the call lineDevSpecifc(CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C C1 Unparks the call Varaint : if LineDevSpecific for receiving Terminated Event is not set	Call between B and C is setup with Secure mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C. Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C. Park Operation is successful and overCallSecurity Status is degraded to Not-Authenticated LINE_REPLY (dwRequestId, 0) is returned UnPark operation is Successful and Monitoring session is terminated. Call on C1 is disconnected as C1doesn't have Secure Capabilities. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C. Terminated Event is not Reported

Silent Monitoring on Conferenced Call

Action	Expected result
LineInitialize Device A and B1 is not Secure Device C and B is Secure LineOpen on A,B,B1 and C A, B and B1 are in Conference C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input	Silent Monitoring Call between B and C is setup with Secure mode. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C. Call Security Status = Not Authenticated

Conference on Monitored Call

Action	Expected result
LineInitialize. Device A, B and C is not Secure Device C1 is Secure LineOpen on A,B,C and D A calls B;B answers the Call C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input C creates conference with C1 LineGetCallInfo on B LineGetCallInfo on C LineGetCallInfo on C1	Monitoring Request is successful and the Session is started Conference is created with A , C and C1 Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C1. Call Security Status = Not Authenticated SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain CFB's info. CallAttributeTye = CallAttribute_SilentMonitorCall Call Security Status = Not Authenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Conference on Monitored Call

Action	Expected result
LineInitialize Device A, B and C is Secure Device C1 is not Secure LineOpen on A,B,C and C1 A calls B;B answers the Call C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input lineDevSpecifc (CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C C creates and Completes conference with C1	Monitoring Request is successful and the Session is started Monitoring Session is ended and C and C1 will be in direct simple call. Line_CallDevSpecific will be fired for B. dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, Dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C

Supervisor Holds the Call

Action	Expected result
LineInitialize Device A, B and C is Secure Device C1 is Secure LineOpen on A,B,C and C1 C and C1 are shared lines A calls B; B answers the Call C issues LineDevSpecific (Start Monitoring) with A's permanent lineID, silent monitoring mode and NoTone as input C holds the call C1 resumes the call Variant: C1 is not Secure and DEVSPECIFIC_SILENT_MONITORING_TERMINATED filter is enabled on C	Monitoring session is started Media will be stopped Media is started.Call on C will be INACTIVE (RIU Call) Monitoring session is Terminated. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Call on C1 will be Disconnected with new Cause Code LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C

Recording

Set up

User is in Allow Recording group

A is Customer Device

B is Agent

C is Recording Device

BIB on B is set to on.

Recording Type on B is Application Invoked

C is configured as the recording device for B

Basic Recording Scenario

Action	Expected result
LineInitialize Device A,B and C is not-Secure LineOpen on A and B A calls B;B answers the Call B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call LineGetCallInfo on B	Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to on B CallReason = LINECALLREASON_DIRECT Devspecific part will contain the following CallAttributeTye = ‘CallAttribute_RecordedCall’ Address = C’s DN, Partition = C’s Partition Device Name = C’s Device Name Transaction ID = 0 Call Security Status = Not Authenticated
Variant 1 : Monitor the Customer and Agent Lines after the Recording Session is Started. LineGetCallInfo on B	CallReason = LINECALLREASON_UNKNOWN

Basic Recording Scenario in Secure Mode

Action	Expected result
LineInitialize Device A,B and C is Secure LineOpen on A and B A calls B;B answers the Call A to B call is Secure B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call LineGetCallInfo on B	Recording session is started in secure mode Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired on B SRTP info will be available (for A-B Call) Devspecific part will contain the following: CallAttributeTye = CallAttribute_RecordedCall Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = 0 Call Security Status = Encrypted

Recording Scenario on Non-Secure Call in Secure Mode

Action	Expected result
LineInitialize Device A is not Secure Device B and C is Secure LineOpen on A and B A calls B;B answers the Call A to B call is non Secure B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call LineGetCallInfo on B	Recording session is started in secure mode Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B SRTP Info is not available Devspecific part will contain the following: CallAttributeTye = CallAttribute_RecordedCall Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = 0 Call Security Status = Not Authenticated

Recording Scenario on Non-Secure Call Using Secure Recording Profile/Device

Action	Expected result
LineInitialize Device A and B is Secure Device C is Not Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Recording Request will Fail with existing error code LINEERR_OPERATIONFAILED **Note** Recording Failed as the Recording Device Security Capabilities doesn't meet or exceed the Security status of B

Recording Scenario When Agent Holds the Call

Action	Expected result
LineInitialize Device A and B is not Secure Device C is Secure LineOpen on A and B A calls B;B answers the Call A to B call is non Secure B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call LineHold on Call on B B resumes the Call **Note** Recording option – Automatic Call Recording Enabled B Resumes the Call	Recording Session is started Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B Call between B and C will be Non-Secure Media between B and C is ended Line_CallDevSpecific (dwparam1 = RecordingEnded) will be fired for B Recording Session will be started Media between B and C is started Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B Recording Session will be started

Recording and Monitoring

This section describes Silent Monitoring and Recording on Agent Call in Secure Mode.

Both Silent Monitoring and Recording on Agent Call in Secure Mode

Action	Expected result
LineInitialize Device A,B,C and D are Secure D is configured as Recording Device on B LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input LineGetCallInfo on B LineGetCallInfo on C	Silent Monitored Call is created in Secure Mode C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as inputLine_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Encrypted SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Action	Expected result
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call LineGetCallInfo on B	Recording session is started in secure mode Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired on B SRTP info will be available (SRTP info for the call Between B and A) Devspecific part will contain the following: CallAttributeTye = ‘CallAttribute_SilentMonitorCall’ Address = C’s DN, Partition = C’s Partition Device Name = C’s Device Name Transaction ID = XXXX CallAttributeTye = CallAttribute_RecordedCall Address = D’s DN, Partition = D’s Partition Device Name = D’s Device Name Transaction ID = 0 Call Security Status = Encrypted

Recording Silent Monitored Call on Supervisor

Action	Expected result
LineInitialize Device A and B is not Secure Device C and D is Secure D is the Recording Device D is configured as Recording on C LineOpen on A, B and C A calls B;B answers the Call A to B call is non Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input LineGetCallInfo on B LineGetCallInfo on C C issues LineDevSpecific (Start Recording, BothLocalAndRemote) for B-C call	Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B New call will be fired on C (Silent Monitoring call) Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Unauthenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated Recording Session is started Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to C

Action	Expected result
LineGetCallInfo on C	SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallAttributeTye = 'CallAttribute_RecordedCall' Address = D's DN, Partition = D's Partition Device Name = D's Device Name Transaction ID = 0 Call Security Status = Not Authenticated

Shared Lines-Initiating a New Call Manually

The following table describes the message sequences for Shared Lines-Initiating a new call manually where Party A and Party A' represent shared line appearances. Also, Party A and Party A' are idle.

Action	CTI messages	TAPI messages	TAPI structures
1. Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct, RIU = false	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change
1. Party A goes off-hook	Party A’		
	NewCallEvent, CH = C1, GCH = G1, Calling = A’, Called = NP, OrigCalled = NP, LR = NP, S tate = Dialtone, Origin = OutBound, Reason = Direct, RIU = true	LINE_APPNEWCALL hDevice = A’ dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-2 dwParam3 = OWNER	LINECALLINFO (hCall-2) hLine = A’ dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A’ dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0	No change

Action	CTI messages	TAPI messages	TAPI structures
2. Party A dials Party B	Party A		
	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	Party A’		
	None	None	None
3. Party B accepts call	Party A		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CalledChanged = true, Called = B, Reason = Direct, RIU = false	Ignored	No change

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLERID, CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change
3. Party B accepts call (continued)	Party A’		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A’, CalledChanged = true, Called = B, Reason = Direct, RIU = true	Ignored	No change

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A', Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CALLERID, CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A' dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A' dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A', Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0	No change
4. Party B answers call	Party A		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0, dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	Party A’		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A’, Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0, dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A’ dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A’ dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

SRTP

Media Terminate by Application (Open Secure CTI Port or RP)

- Negotiate version
- Sends LineOpen with extension version as 0x8007000
- Send CciscoLineDevSpecificUserSetSRTPAlgorithmID
- Send CCiscoLineDevSpecificUserControlRTPStream
- Now, the CTI port or RP gets registered as secure port
- Make call from secure IP phone to the CTI port or RP port
- Answer the call from application
- SRTP indication gets reported as LineDevSpecific event
- SRTP key information get stored in LINECALLINFO::devSpecifc for retrieval

Media Terminate by TSP Wave Driver (Open Secure CTI Port)

- Negotiate version
- Sends LineOpen with extension version as 0x4007000
- Send CciscoLineDevSpecificUserSetSRTPAlgorithmID
- Send CciscoLineDevSpecificSendLineOpen

- Now, the CTI port gets registered as secure port
- Make call from secure IP phone to the CTI port
- Answer the call from application
- SRTP indication gets reported as LineDevSpecific event
- SRTP key information get stored in LINECALLINFO::devSpecifc for retrieval

Support for Cisco IP Phone 6900 Series

Use cases related to Cisco Unified IP Phone 6900 Series support feature are mentioned below:

Monitoring Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 Phone with Roll Over Mode User is added to New User Group. Application does Line Initialize
Expected Results	Lines on the Cisco Unified IP Phone 7931 will be enumerated. Application would be able to Open Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 and it would be able to control and perform call operations on phone.

Monitoring Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode Step 1: Application does Line Initialize Step 2: User is added to New User Group.
Expected Results	Step 1: Lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 will not be enumerated Application will not be notified about the device A and it will not be able to monitor. Step 2: Application will be receiving PHONE_CREATE and LINE_CREATE events for the Device and lines on that Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode. Now Applications would be able to Monitor and control Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931.

Transfer Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to new user Group.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A. Variants: Application Opens only Line A on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931
Expected Results	Call on A will go to OnHold State. New call will be created on Line B. Application then has to complete Transfer using DTAL feature. Variants: Applications would not be able to Complete Transfer from Application as the Line B is not monitored.

Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Conference scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group.

Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D are two SCCP phones Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize C calls A,A answers SetupConference on A.
Expected Results	Call on A will go to OnHold State. New call will be created on Line B. Application then has to complete Conference using Join Across Lines feature.

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/OnHoldPendingConference. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with Conference LineCompleteTransfer with Mode as Conference to complete Conference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -Roll Over to any Line Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/OnHoldPendingConference. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with Conference LineCompleteTransfer with Mode as Conference to complete Conference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.

Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	SetupTransfer Request will fail with error "LINEERR_CALLUNAVAIL".
Variants	Test the same Scenario with SetupConference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/Conference State. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with SetupConference

LineMakeCall Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing LineMakeCall Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN" or "Roll Over to Any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers LineMakeCall on A.
Expected Results	LineMakeCall Operation will fail with error "LINEERR_CALLUNAVAIL". Roll Over Doesn't Happen to second line as the roll over is only for Outbound Calls.

LineUnPark Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing LineUnPark Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN" or "Roll Over to Any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers LineUnPark on A.(tires to retrieve the available Parked Call from Park DN)

Expected Results	LineUnPark Operation will fail with error "LINEERR_CALLUNAVAIL". Roll Over Doesn't Happen to second line as the roll over is only for Outbound Calls.

EM Login/Logout Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing EM Log In/Out Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. EM Profile is logged onto the Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Test the Use Case from UseCase#1 to UseCase#10
Expected Results	Same as the Use Case tested.

Manual Transfer Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Existing Call Events on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -Roll Over to any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press Transfer Button on Cisco Unified IP Phone 6900 Series and Dial D. Step 4: Answer the Call on D Step 5: Complete Transfer from Phone A Variant: Monitor Phones after Transfer is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Call will be on ONHOLDPENDINGTRANSFER state. Step 5: Both the calls on A and B will go to IDLE state. C and D will be in Simple Call. Variant: Same as this Use Case

Manual Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press conference Button on Cisco Unified IP Phone 6900 Series and Dial D. Step 4: Answer the Call on D Step 5: Complete Conference from Phone Variant: Monitor Phones after Conference is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Conference Model is created as today on Non-Cisco Unified IP Phone 6900 Series. Step 5: A ,C and D will be in conference Conference model will be created on Line A. Variant: Same as this Use Case.

Manual Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press conference Button on Cisco Unified IP Phone 6900 Series Phone and Dial D. Step 4: Answer the Call on D Step 5: Complete Conference from Phone Variant: Monitor Phones after Conference is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Conference Model is created as today on Non-Cisco Unified IP Phone 6900 Series Phone. Step 5: A ,C and D will be in conference Conference model will be created on Line A. Variant: Same as this Use Case.

SetupConference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger : 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers Step 1: SetupTransfer on A. Step 2: Complete Conference From Phone.
Expected Results	Step 1: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. Step 5: A ,C and D will be in conference Conference model will be created on Line A.

BWC on Cisco Unified IP Phone 7931 in Non Roll Over Mode When User Is Removed From New User Group

Description	Testing Cisco Unified IP Phone 7931 Phone behavior in Non Roll Over Mode When User is removed from New User Group.

Test Setup	User is Removed from New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Non-Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Non Roll Over Mode" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize
Expected Results	Lines on the Cisco Unified IP Phone 7931 will be enumerated. Application would be able to Open Cisco Unified IP Phone 7931 with Non-Roll Over Mode and it would be able to control and perform call operations on Phone.

Acquire Device on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Behavior of Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 on Super Provider when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode User is Added to New User Group. Step 1: Application does Line Initialize Step 2: LineDevSpecific to Acquire Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 3: User is removed from New User Group.
Expected Results	Step 2: Application will be receiving PHONE_CREATE and LINE_CREATE events for the Device and lines on that Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode. Step 3: Application will be receiving LINE_REMOVE and PHONE_REMOVE for the Cisco Unified IP Phone 7931 and Application will no longer be able to monitor or control that device.

Support for Cisco Unified IP Phone 6900 and 9900 Series Use Cases

The use cases related to Support for Cisco Unified IP Phone 6900 and 9900 Series are provided below:

Check Max Calls Information

Action	Events, Requests, and Responses
Application calls LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks Max Calls field.	MaxCalls = 4 in LineDevCaps:DevSpecific

Check Busy Trigger Information

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks busy trigger field.	BusyTrigger = 2 in LineDevCaps:DevSpecific

Check Line Instance

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks line instance field.	LineInstanceNumber = 1 in LineDevCaps:DevSpecific

Check Line Label

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks line label field.	LineLable = label_2000 in LineDevCaps:DevSpecific

Check Voice Mail Pilot

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	VoiceMailPilot = 5000 in LineDevCaps:DevSpecific

Check Registered IP Address of the Device or Line

.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks IP address field.	LineInitialize successful RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check IP address field.	PhoneInitialize successful RegisteredIPv4Address & RegisteredIPv6Address available in PhoneDevCaps:DevSpecific

Check Consult Rollover Information of the Line

ConsultRollOver is true for the device

.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks consult roll over field.	LineInitialize successful ConsultRollOver flag is true in LineDevCaps:DevSpecific
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful ConsultRollOver flag is true in PhoneDevCaps:DevSpecific.
Variance: Phone does not support rollover Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful ConsultRollOver flag is false in PhoneDevCaps:DevSpecific.

Check JAL or DTAL Information of the Line

JAL or DTAL is true for the device.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks JAT/DTAL field.	LineInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is true in LineDevCaps:DevSpecific.
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is true in PhoneDevCaps:DevSpecific.
Variance: Phone does not support jal/dtal Perform PhoneInitialize and check PhoneGetDevCpas to check JAT/DTAL field.	PhoneInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is false in PhoneDevCaps:DevSpecific.

Handle Voice Mail Pilot Change

Voice Mail Pilot number is changed to 6000.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	LineInitialize successful VoiceMailPilot = 5000 in LineDevCaps:DevSpecific
Voice Mail Pilot number is changed to 6000.	LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating Voice Mail Pilot is changed.
Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	VoiceMailPilot = 6000 in LineDevCaps:DevSpecific
Variance: also applies to Line Label	

Check IP Address When Device Is Unregistered or Registered

It is assumed that phone uses static IP address and is already registered.

Action	Events, Requests, and Responses
Application calls LineInitialize Application calls LineGetDevCaps, and checks IP address field.	Initializesuccessful RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific, and RegisteredIPAddressMode is IPAddress_IPv4_IPv6.
Reset device	Phone or line goes out of service. LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address information is changed.
Application calls LineGetDevCaps, and checks IP address field.	The same RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific, but RegisteredIPAddressMode is IPAddress_Unknown.
Device re-registered with CUCM.	Phone or line back in service. LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address information is changed.
Application calls LineGetDevCaps, and checks IP address field.	The same RegisteredIPv4Address and RegisteredIPv6Address available in LineDevCaps:DevSpecific, but RegisteredIPAddressMode is set to IPAddress_IPv4_IPv6.
Variance: Phone uses DHCP and new IP address is obtained for registering.	LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address is changed New IPAddress will be in devSpecific when application queries LineGetDevCap. .

Swap or Cancel

Use cases related to Swap or Cancel feature are mentioned below:

Connected Transfer

Device A, B, C where A is a Cisco Unified IP Phone (future version)..

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press transfer	For A: Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-3 DIALTONE
A picks "Active Calls"	Call-3 goes IDLE

Action	Expected events
A picks call (A‡C) and presses transfer to complete transfer	For A: Both calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Connected Transfer on Phones with Shared Lines

Device A, B, C, A' where A and A' are sharedline.

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A For A': Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED_INACTIVE Caller = A, Called = B Connected B

Action	Expected events
User performs connected transfer on Cisco Unified IP phone (future version)	For A and A': All calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Connected Transfer: Initiate From Phone, Complete From CTI

Device A, B, C .

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
Application sends either CompleteTransfer or DirectTransfer on A	For A and A': All calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Consult Transfer: Resume Primary Call (Implicit Cancel)

Action	Expected events
A ‡ B A setup consult transfers to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press resume to resume A‡ B call	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Consult Transfer: Swap Calls

Action	Expected events
A ‡B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C

Action	Expected events
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press "Transfer" to complete transfer	For A: Calls go IDLE For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Consult Transfer on Phone: Swap Calls; CTI Sends SetupTransfer on Connected Call

Action	Expected events
A ‡ B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
Application calls LineSetupTransfer on A's connected call (A‡B) to initiate transfer	Request succeeds as phone cancels existing feature plan and allow CTI request to go through.

Consult Transfer: Swap and Cancel

Action	Expected events
A ‡ B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses Cancel	No TSP event since it is handled during swap operation

RoundTable Connected Conference

Action	Expected events
A ‡ B A puts call on hold A creates new call to C, C answer	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Conference"	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGCONFENRENCE Caller = A, Called = C Connected C Call-3 DIALTONE

Action	Expected events
A picks active call (A‡ C) on phone UI, and presses "Conference" to complete the conference	For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = C, connected = C Call-3 IDLE For B: For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = C, connected = C For C: For A: CONNECTED CONFERENCED Caller = A, called = C, connected = C CONFERENCED Caller = C, called = B, connected = B

RoundTable Connected Conference: Cancel

Action	Expected events
A ‡ B A puts call on hold A creates new call to C, C answers	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Conference"	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONFERENCED Caller = A, Called = C Connected C Call-3 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGCONFENRENCE Caller = A, Called = C Connected C Call-4 DIALTONE

Action	Expected events
A picks "Active Calls"	For A: Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-3 / Call-4 IDLE
A presses Cancel softkey	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Set Up Consult Conference From RT, Then Swap and Complete Conference From RT

Action	Expected events
A ‡ B A sets up conference to C, C answer	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Swap"	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = C Connected C

Action	Expected events
A presses "Conference" to complete conference	For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = C, connected = C For B: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = C, connected = C For C: For A: CONNECTED CONFERENCED Caller = A, called = C, connected = C CONFERENCED Caller = C, called = B, connected = B

Set Up Consult Conference From RT, Then Swap and Cancel From Phone with Shared Line Scenario

A and A’ are shared lines..

Action	Expected events
A ‡ B A sets up conference to C, C answers	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C For A' CONNECTED INACTIVE Caller = A, celled = B, connected = B CONNECTED INACTIVE Caller = A, celled = C, connected = C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Swap"	For A: The scenario looks the same when primary call resumes Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C

Action	Expected events
A presses "Cancel"	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected = C
	For A' Call-1 LINE_CALLSTATE CONNECTED INACTIVE Caller = A, Called = B Connected = B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Set Up Consult Conference From RT: Resume Primary Call (Implicit Cancel)

Action	Expected events
A ‡ B A sets up conference to C, C answer	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C
	For A' CONNECTED INACTIVE Caller = A, celled = B, connected = B CONNECTED INACTIVE Caller = A, celled = C, connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A resumes A‡B call	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

User Is Removed From Standard Supports Connected Xfer/Conf Group

Action	Expected events
User is in Standard Supports Connected Xfer/Conf group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP
Remove user from "Standard Supports Connected Xfer/Conf" user group	APP receives PHONE_REMOVE / LINE_REMOVE

User Is Removed From Standard Supports Connected Xfer/Conf Group

Action	Expected events
User is in Standard Supports Connected Xfer/Conf group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP

Action	Expected events
Remove user from Standard Supports Connected Xfer/Conf user group	APP receives PHONE_REMOVE / LINE_REMOVE

User Is Removed From Standard Supports Connected Xfer/Conf Group While Line Is Open

Action	Expected events
user is in "Standard Supports Connected Xfer/Conf" group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP
App sends LineOpen to open line on Cisco Unified IP phone (future version) phone	Successful
Remove user from Standard Supports Connected Xfer/Conf group	TSP sends LINE_CLOSE APP receives LINE_REMOVE

User Is Added to Standard Supports Connected Xfer/Conf Group

Action	Expected events
user is not in "Standard Supports Connected Xfer/Conf" group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is not enumerated to APP
Add user to Standard Supports Connected Xfer/Conf group	APP receives PHONE_CREATE / LINE_CREATE

Unrestricted Unified CM

Table 106: Application Tries Secure Connection to Unrestricted Unified CM During Upgrade

Action	Events, requests and responses
CUCM – Restricted UCM TSP is configured to connect Secure Application calls LineInitialize *** Upgrade CUCM to Unrestricted Unified CM CCM/CTI services restarted	LineInitialize successful All lines associated are enumerated. OutOfService Events for all the Devices/Lines. ***TSP will internally try to Connect CTI in Secure mode. As CTI is upgraded to Non-secure, the Connection Fails and applications are not notified. Application has to disable “Secure Connection to CTI Manager” on the Security tab in TSP UI to setup connection to CTI/CUCM.

Table 107: Application Tries Secure Connection to Unrestricted Unified CM After Upgrade

Action	Events, requests and responses
CUCM – Restricted UCM TSP is configured to connect Secure Application calls LineInitialize Application calls LineShutdown *** Upgrade CUCM to Unrestricted UCM Application calls LineInitialize	LineInitialize successful All lines associated are enumerated. LineShutDown successful LineInitialize successful. No lines are enumerated to application.

Table 108: Registering Secure CTI Port with Unrestricted Unified CM CTI Manager

Action	Events, requests and responses
CUCM – Unrestricted UCM Setup Non-Secure Connection Application calls LineInitialize Register CTI Port in Secure Mode • LineOpen – with Ext – 80070000 • LineDevspecific – CciscoLineDevSpecificUserSetSRTPAlgorithmID	LineInitialize successful All lines associated to end users are enumerated. LineReply – with error -LINEERR_OPERATIONUNAVAIL

Table 109: Registering Secure CTI Port with Unrestricted Unified CM CTI Manager

Action	Events, requests and responses
Setup: • Node 1 – UnRestricted UCM • Node 2 – Restricted UCM – Secure CTI Port – Device Pool – with Node 1 as High Priority CM. TSP is configured to connect to CTI Manager of Node 2. Set up Secure Connection Application calls LineInitialize Register CTI Port in Secure Mode • LineOpen – with Ext – 80070000 • LineDevspecific – CciscoLineDevSpecificUserSetSRTPAlgorithmID • LineDevSpecific -CCiscoLineDevSpecificUserControlRTPStream	LineInitialize successful All Lines Associated are Enumerated. LineReply – success LINE_CLOSE for the CTI Port

LineHold Enhancement

Prerequisites

Pre-conditions to all persistent call use cases, unless specified otherwise:

- Device A (IP Phone, Line A1 (dn: 1000))
- Device B (IP Phone, Line B1 (dn: 2000))
- 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
- Provider is opened (lineInitializeEx successfully executed)
- All relevant lines are opened with Extension version 0x000D0000 and in service

Table 110: Basic Case - Hold with ContentID to Be Played

Action	TAPI Messages	TAPI Structures
Create Call: LineMakeCall() on Line-A w ith DestAddress="DN of B" and B answers the Call	At A: LINE_CALLSTATE dwParam1 = 0x00000100 (CONNECTED) At B: LINE_CALLSTATE dwParam1 = 0x00000100 (CONNECTED)	CallInfo on A: CallerID: 1000 CalledID: 2000 ConnectedID: 2000
Application issues CCiscoLineDevSpecificHoldEx with ContentID = contentID1 on hCall1(call on A1) *** Call will be placed on Hold and VoH stream selected is played to B.	At A: LINE_CALLSTATE dwParam1 = 0x00000400 (LINECALLSTATE_ONHOLD)	

Whisper Coaching

Setup

Customer Phone – IP Phone A

Agent Phone – IP Phone B

Supervisor Phone – IP Phone C

Application monitoring all lines on all devices

New extension is negotiated when application opens lines

Application Initiates a Whisper Coaching Session

Service Parameter Setting: Observed Target = false, Observed Connected Parties = true

Table 111: Application Initiates a Whisper Coaching Session

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A
C issues CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_LocalOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly **Note** Media events are not received at B. At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Application Updates the Monitoring Mode

Service Parameter Setting: Observed Target = true, Observed Connected Parties = false

Table 112: Application Updates the Monitoring Mode (Silent to WhisperCoaching) and Then Updates the Monitoring Mode (WhisperCoaching to Silent)

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A
C issues CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_RECEPTION)

Action	Events, Requests, and Responses
C issues CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_BothLocalAndRemote	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Whisper_Coaching, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Whisper_Coaching, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
C issues CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Silent tone = PlayToneDirection_NoLocalOrRemote	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Agent Holds the Customer Call with Whisper Coaching Then Agent S Shared Line Resumes the Call

Additional Setup: Agent shared line IP Phone B

Table 113: Agent Holds the Customer Call with Whisper Coaching, Then Agent's Shared Line Resumes the Call

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly B holds the call	At B: ONHOLD devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At B': ONHOLD At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_STOP_RECEPTION)
B resumes the call	At B: CONNECTED At B': CONNECTED, INACTIVE At C: LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Action	Events, Requests, and Responses
B holds the call B resumes the call	At B: CONNECTED, INACTIVE LineDevSpecific(SLDSMT_MONITORING_ENDED) At B': CONNECTED At C: IDLE

Agent Transfers a Whisper Coaching Call Monitoring Call Goes Idle at the Supervisor

Additional Setup: IP Phone D

Table 114: Agent Transfers a Whisper Coaching Call, Monitoring Call Goes Idle at the Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly B setup transfer to D and D answers	At B: ONHOLDPENDTRANSFER devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = B, Called = D, Connected = D At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
B complete transfer to D	At B: IDLE IDLE At C: IDLE

Application Updates the Monitoring Mode (WhisperCoaching to Silent)

Additional Setup: IP Phone D

Table 115: Application Updates the Monitoring Mode (WhisperCoaching to Silent) After the Agent Conferences the Whisper Coaching Call

Action	Events, Requests, and Responses
A initiates Call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly B setup conference to D and D answers B complete conference to D	At B: CONFERENCE Calling = A, Called = B, Connected = B CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONFERENCE Calling = B, Called = D, Connected = D At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
C issues a CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONFERENCE Calling = A, Called = B, Connected = B CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONFERENCE Calling = B, Called = D, Connected = D At C: LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
B issues a lineRemoveFromConference to drop D.	At B: CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly IDLE IDLE At C: No change in callInfo and no additional events

Supervisor Holds/Resumes the Whisper Coaching Monitoring Session

Additional Setup: IP Phone D

Table 116: Supervisor Holds/Resumes the Whisper Coaching Monitoring Session

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C holds the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: ONHOLD Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_STOP_RECEPTION)
C resumes the call	At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Supervisor Transfers the Whisper Coaching Session to Another Supervisor

Additional Setup: Supervisor IP Phone D

Table 117: Supervisor Transfers the Whisper Coaching Session to Another Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C setup transfers the call to D, D answers	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: ONHOLDPENDTRANSFER Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONNECTED Calling = C, Called = D, Connected = C

Action	Events, Requests, and Responses
C complete transfers the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = D, partition = D's Partition, deviceName = D's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: IDLE IDLE At D: CONNECTED Calling = C, Called = D Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Supervisor Conferences the Whisper Coaching Session to Another Supervisor

Additional Setup: Supervisor IP Phone D

Table 118: Supervisor Conferences the Whisper Coaching Session to Another Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C setup conferences the call to D and D answers	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONFERENCE ONHOLDPENDCONF Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONNECTED Calling = C, Called = D, Connected = C

Action	Events, Requests, and Responses
C complete conferences the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = CFB, partition = CFB Partition, deviceName = CFB device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONFERENCE Calling = C, Called = B/B’s Name, Connected = CFB CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B’s Partition, deviceName = B’s device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONFERENCE Calling = C, Called = D, Connected = D CONNECTED CONNECTED Calling = D, Called = CFB, Connected = CFB

Action	Events, Requests, and Responses
C drops the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = D, partition = D's Partition, deviceName = D's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: IDLE IDLE IDLE At D: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = B, partition = B's Partition, deviceName = B's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly
D issues a CciscoLineDevSpecificMonitoringUpdateMode with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	

Application Initiates a Whisper Coaching Session Second Application on a Different Client Opens All Lines

Additional Setup: Supervisor IP Phone D

Table 119: Application Initiates a Whisper Coaching Session, Second Application on a Different Client Opens All Lines

Action	Events, Requests, and Responses
A initiates Call to B, B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly	At B (Application 1): CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C (Application 1): CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
Second application opens all lines	At B (Application 2): CONNECTED devSpecific CallAttributeBitMask = TSPCallAttribute_WhisperMonitorCall type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C (Application 2): CONNECTED CallAttributeBitMask = TSPCallAttribute_WhisperMonitorCall_Target Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Secure R & M with Whisper Coaching Supports

- Overall security status of the monitoring call either silent or whisper must be same. See Secure monitoring use cases.
- Overall security status of the monitoring call must not change if monitor mode is updated either from silent to whisper or vice versa.

Application Initiates a Secure Whisper Coaching Session

Additional Setup: All devices are secure

Table 120: Application Initiates a Secure Whisper Coaching Session

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A

Action	Events, Requests, and Responses
C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_LocalOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CallSecurityStatus = OverallCallSecurityStatus_Encrypted **Note** Media events are not received at B and SRTP keys are not available. At C: LineDevSpecific (dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) SRTP keys are available CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CallSecurityStatus = OverallCallSecurityStatus_Encrypted LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Application Updates the Monitoring Mode on an Agent Call That Is on Hold

The application updates the monitoring mode on an agent call that is on hold as follows:

1. A initiates Call to B and B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:

- permLineId = B permLineId
- mode = MonitorMode_Whisper_Coaching
- tone = PlayToneDirection_RemoteOnly

3. B puts the call on hold
4. C issues CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
5. LINE_REPLY returns LINEERR_INVALCALLSTATE

Application Initiates Whisper Coaching Where the Agent Is a SIP Device with Older Firmware Version That Does Not Support Media Mixing

The application initiates Whisper Coaching where the agent is a SIP device with older firmware version that does not support media mixing as follows:

1. A initiates Call to B and B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
3. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

Application Updates the Monitoring Mode Where the Agent Is a SIP Device with Older Firmware Version That Does Not Support Media Mixing

The application updates the monitoring mode where the agent is a SIP device with older firmware version that does not support media mixing as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
3. C issues a CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly

4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

Application Updates the Monitoring Mode on a Monitoring Call at the Supervisor That Is in a Conference

The application updates the monitoring mode on a monitoring call at the supervisor that is in a conference as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
3. C setups or completes the call to D and D answers.
4. C issues a CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
5. LINE_REPLY returns LINEERR_OPERATIONUNAVAIL

Application Initiates Whisper Coaching on an Agent That Is Already Playing an Agent Greeting

The application initiates Whisper Coaching on a agent that already is playing an agent greeting as follows:

1. A initiates Call to Band B answers
2. B issues a CCiscoLineDevSpecificStartSendMediaToBIBRequest with:
   - DN = IVR DN
   - timeout = 30
3. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

Application Initiates Agent Greeting on a Call That Already Has a Whisper Coaching Session

The application initiates Agent Greeting on a call that already has a Whisper Coaching session as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:

- permLineId = B permLineId
- mode = MonitorMode_Whisper_Coaching
- tone = PlayToneDirection_RemoteOnly

3. B issues a CCiscoLineDevSpecificStartSendMediaToBIBRequest with:
   - DN = IVR DN
   - timeout = 30
4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL