APPENDIX A

メッセージ シーケンスの図

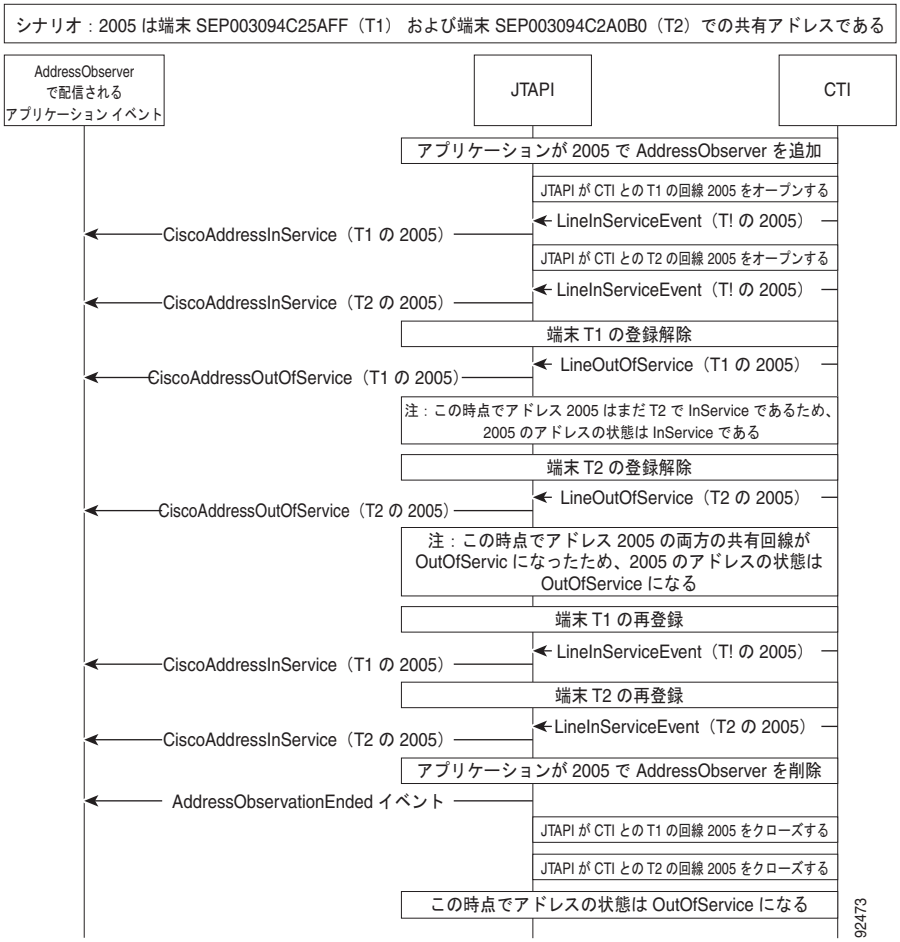
この付録には、次のシナリオに関するメッセージ フローを示すメッセージ シーケンスの図があります。

- 共用回線のサポート
- 転送と直接転送
- 会議と参加
- 割り込みとプライバシー
- CallSelect と UnSelect
- コールごとの CTIPort 動的登録
- ルート ポイントでのメディア終端
- リダイレクト時のオリジナルの着信者番号
- シングル ステップ転送
- 発信側番号の変更
- CTIPort および RoutePoint での AutoAccept
- Forced Authorization Code と Customer Matter Code
- スーパー プロバイダーのメッセージ フロー
- スーパープロバイダーと変更通知の拡張の使用例
- QSIG パス置換
- デバイス ステート サーバ
- パーティションのサポート
- ヘアピン サポート
- QoS のサポート
- TLS セキュリティ
- SRTP 鍵情報
- デバイスと回線の制限
- SIP のサポート
- SIP REPLACE
- Unicode のサポート
- 下位互換性に関する機能拡張
- 半二重メディア

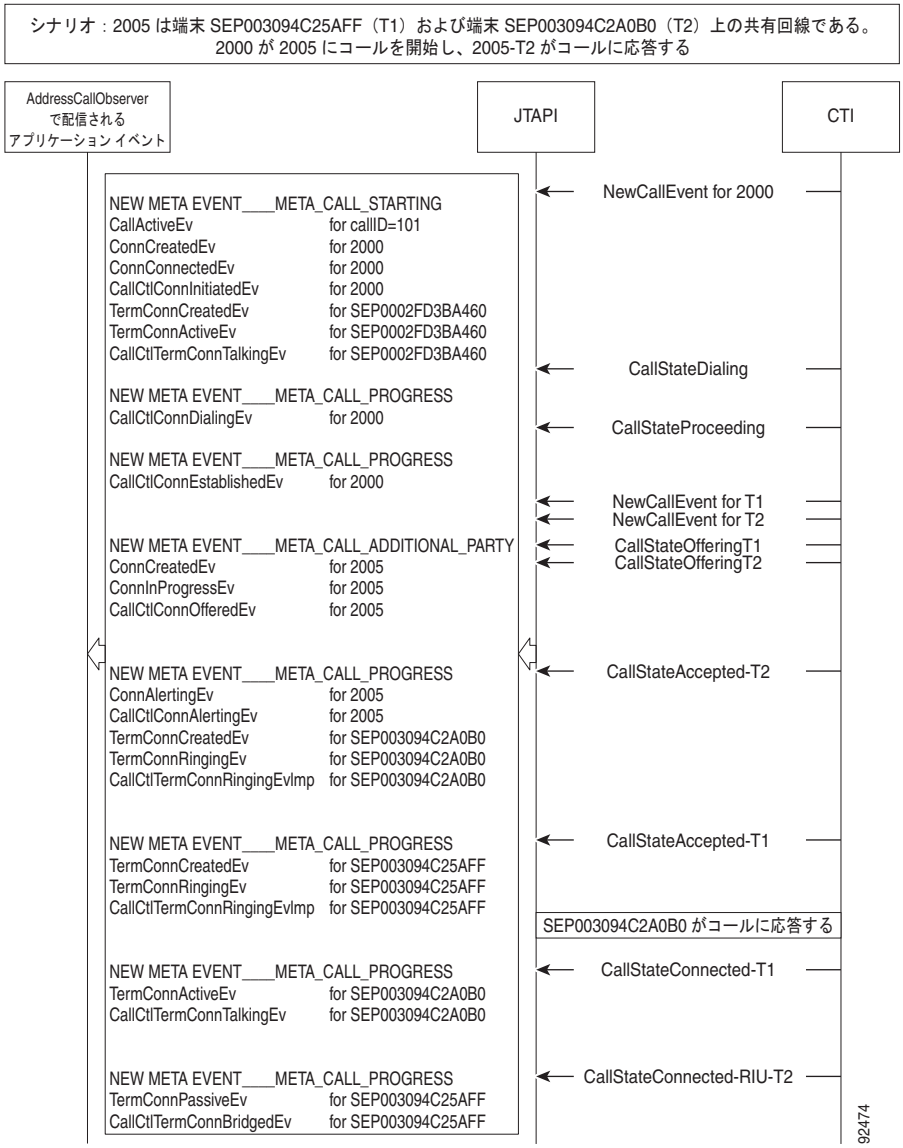
共用回線のサポート

共用回線のサポートに関するメッセージ フローを次に示します。

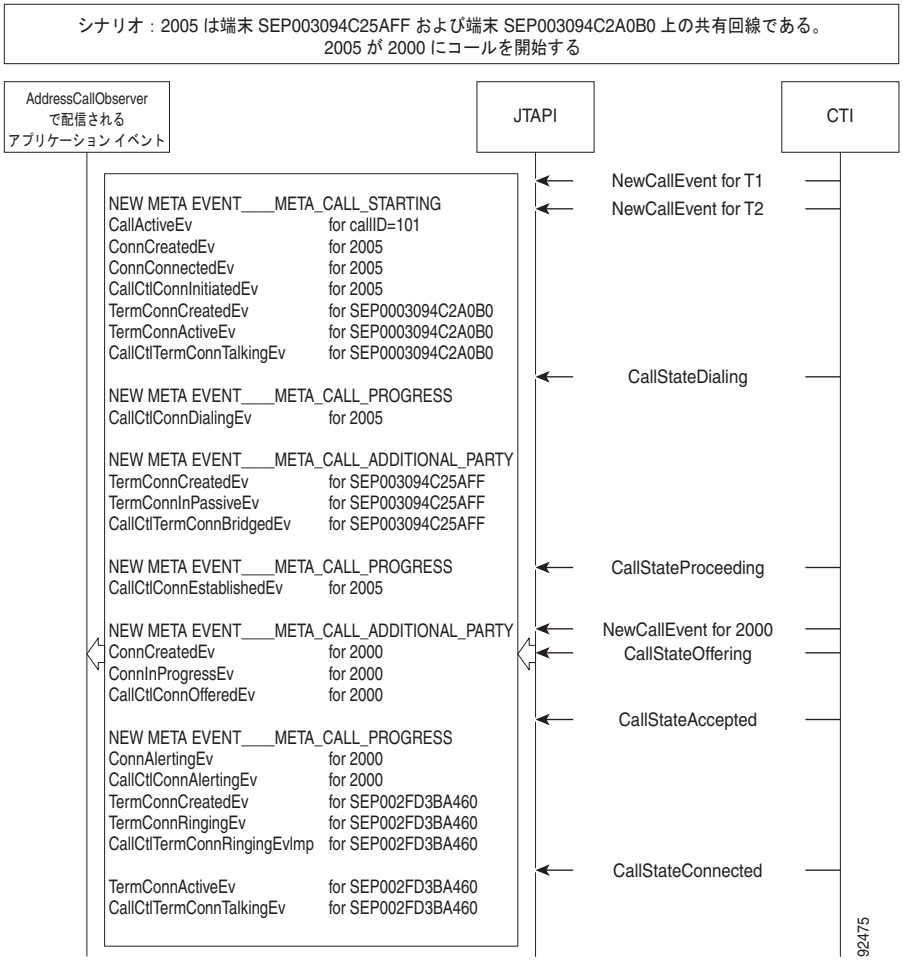
AddressInService/AddressOutofService イベント



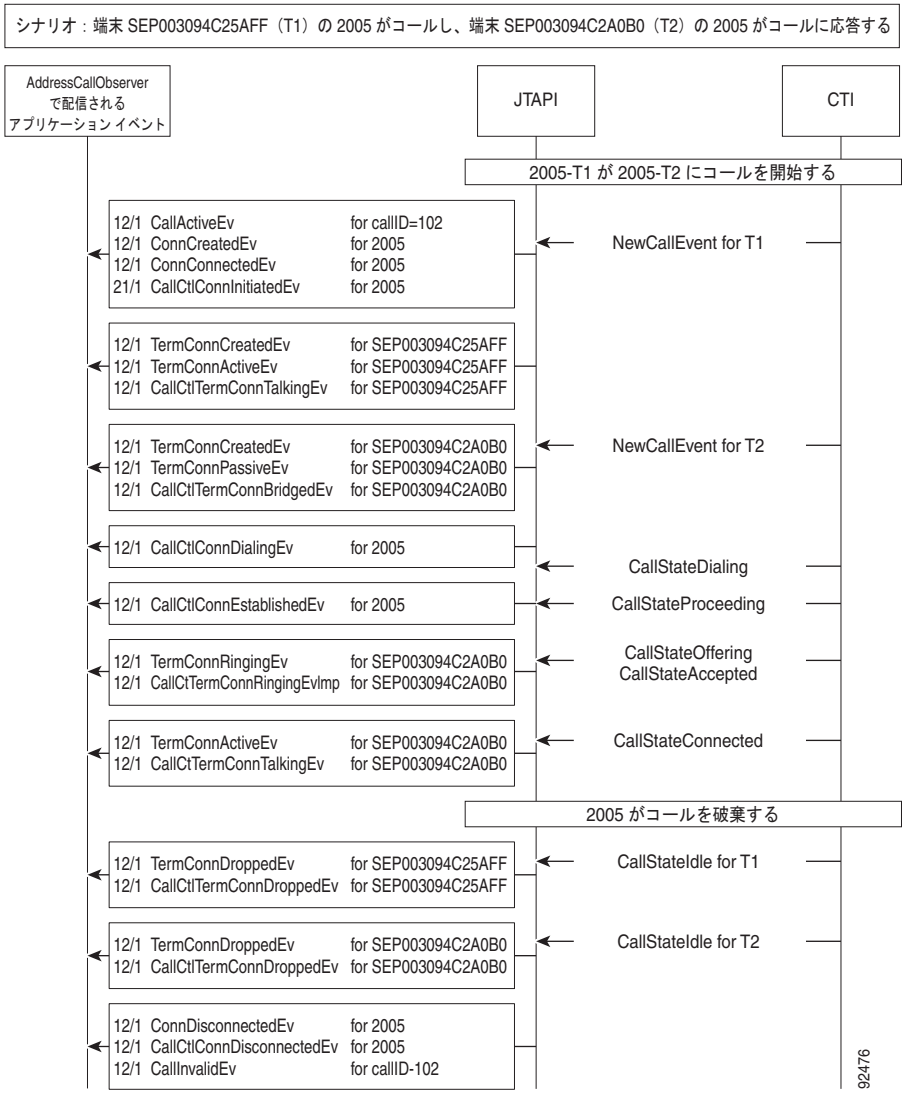
共用アドレスへの着信コール



共用アドレスからの発信コール



共用アドレス自体へのコール

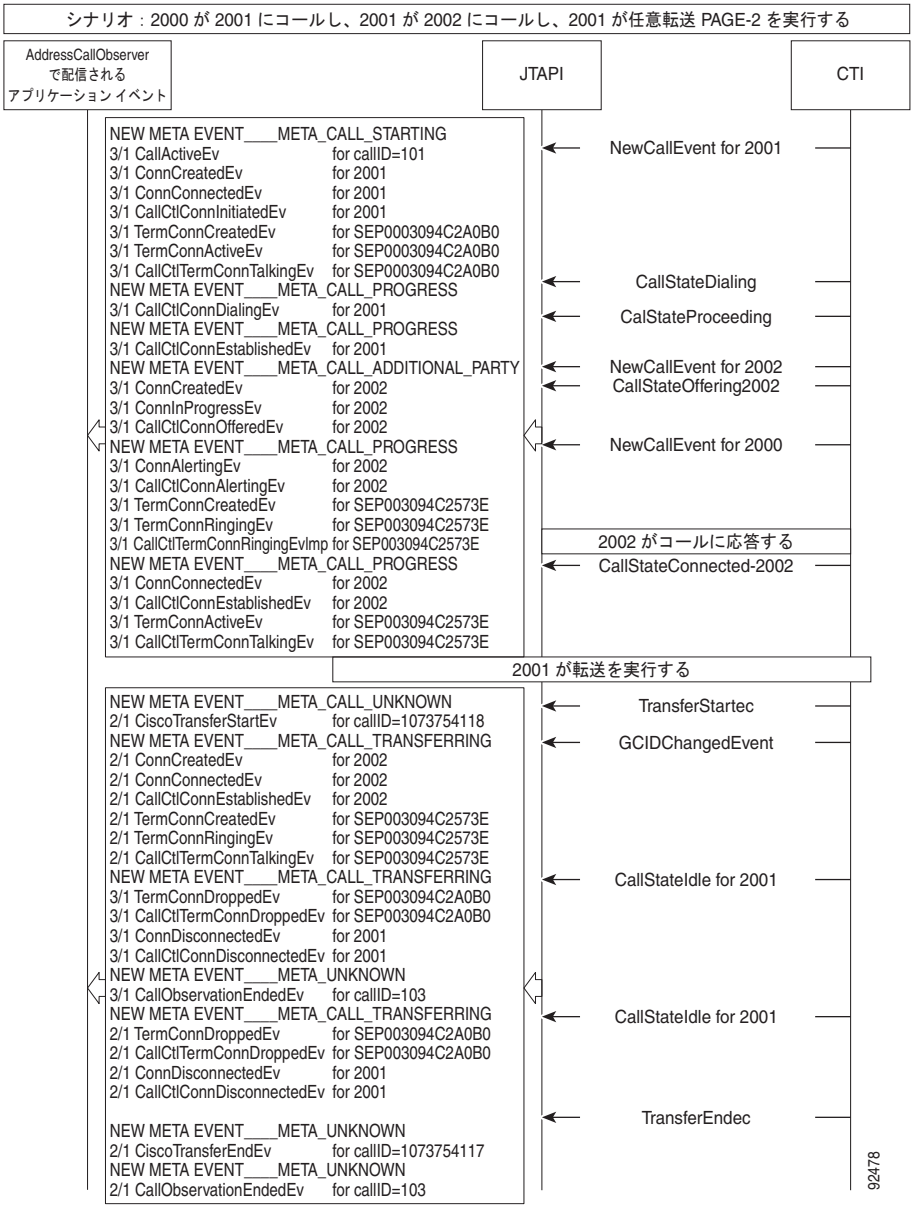


転送と直接転送

転送と直接転送に関するメッセージ フローを次に示します。

直接転送 / 任意転送のシナリオ

直接転送 / 任意転送のシナリオ：ページ 2



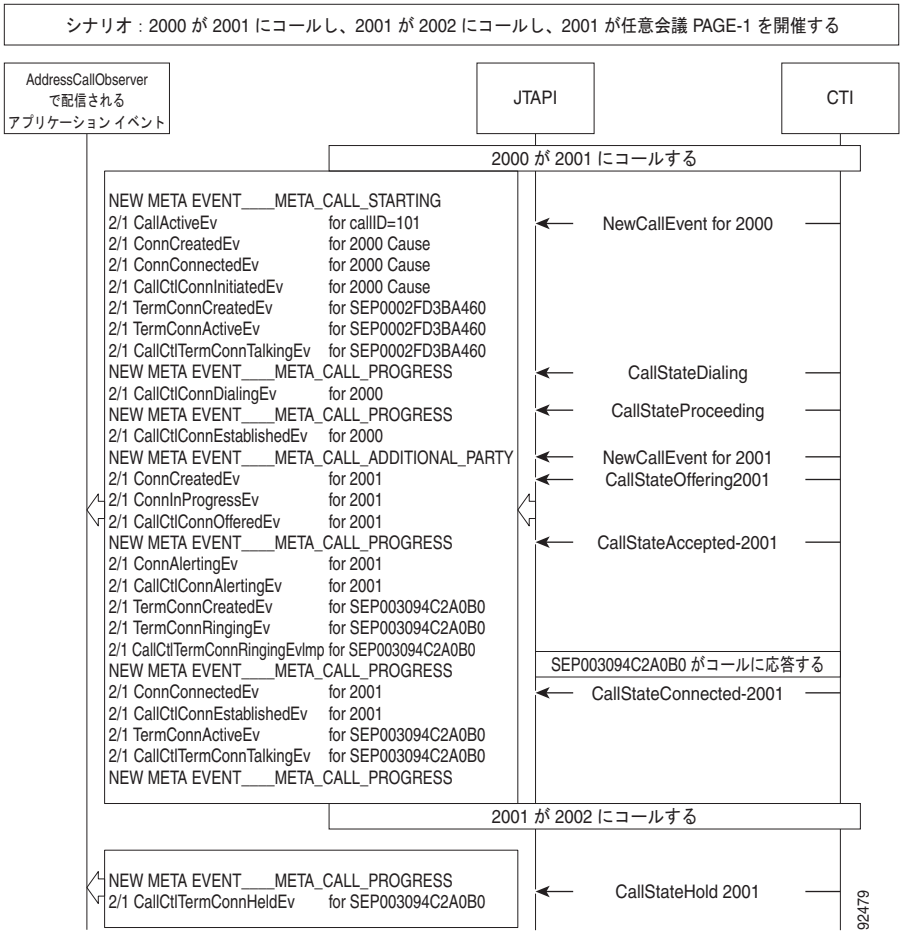
コンサルト転送

コンサルト転送のメッセージ フローは、任意転送のフローと同じです。

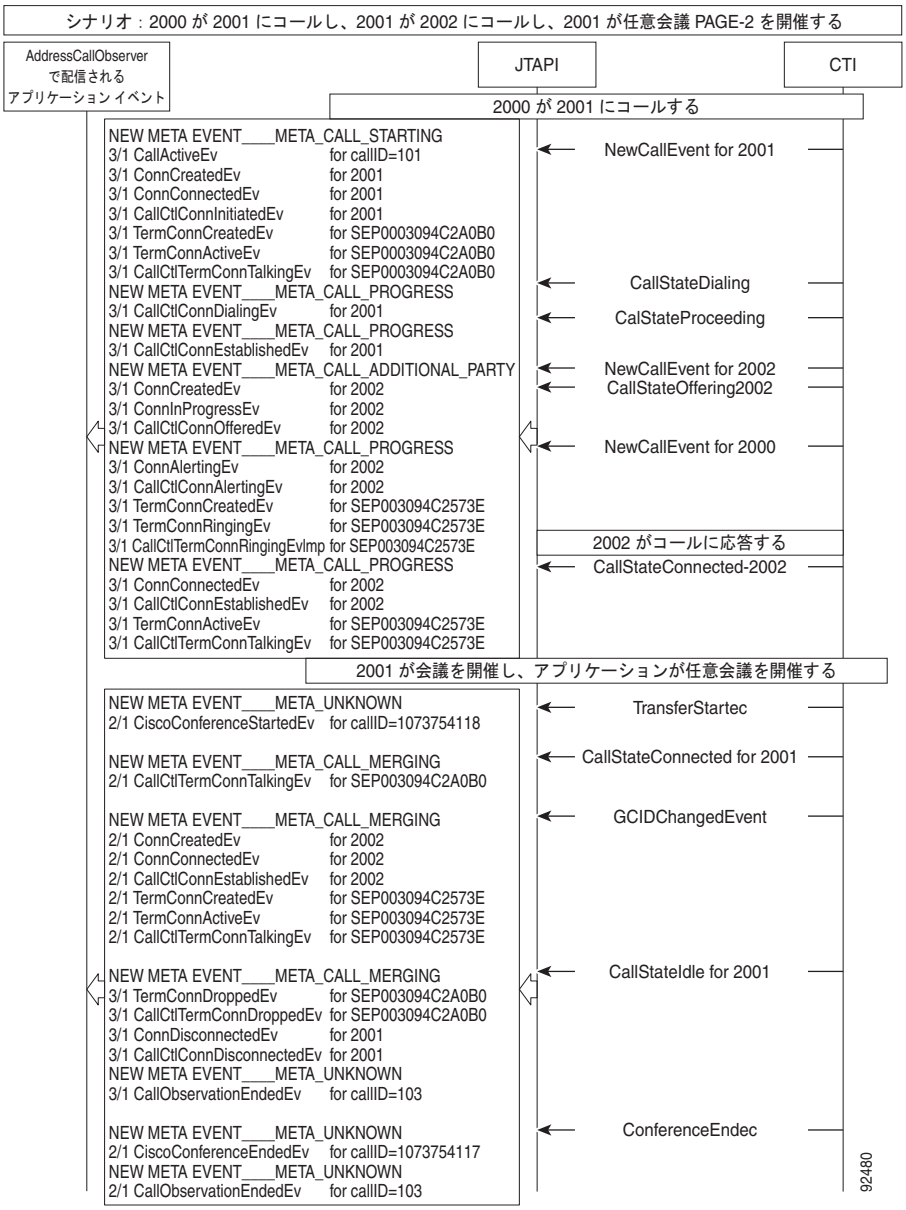
会議と参加

会議と参加に関するメッセージ フローを次に示します。

参加 / 任意会議


シナリオ : 2000 が 2001 にコールし、2001 が 2002 にコールし、2001 が任意会議 PAGE-1 を開催する
AddressCallObserver で配信される アプリケーション イベント
JTAPI
CTI
2000 が 2001 にコールする
NEW META EVENT____META_CALL_STARTING
2/1 CallActiveEv for callID=101
NewCallEvent for 2000
2/1 ConnCreatedEv for 2000 Cause
2/1 ConnConnectedEv for 2000 Cause
2/1 CallCtlConnInitiatedEv for 2000 Cause
2/1 TermConnCreatedEv for SEP0002FD3BA460
2/1 TermConnActiveEv for SEP0002FD3BA460
2/1 CallCtlTermConnTalkingEv for SEP0002FD3BA460
NEW META EVENT____META_CALL_PROGRESS
CallStateDialing
2/1 CallCtlConnDialingEv for 2000
NEW META EVENT____META_CALL_PROGRESS
CallStateProceeding
2/1 CallCtlConnEstablishedEv for 2000
NEW META EVENT____META_CALL_ADDITIONAL_PARTY
NewCallEvent for 2001
2/1 ConnCreatedEv for 2001
CallStateOffering2001
2/1 ConnInProgressEv for 2001
2/1 CallCtlConnOfferedEv for 2001
NEW META EVENT____META_CALL_PROGRESS
CallStateAccepted-2001
2/1 ConnAlertingEv for 2001
2/1 CallCtlConnAlertingEv for 2001
2/1 TermConnCreatedEv for SEP003094C2A0B0
2/1 TermConnRingingEv for SEP003094C2A0B0
2/1 CallCtlTermConnRingingEvImp for SEP003094C2A0B0
SEP003094C2A0B0 がコールに応答する
NEW META EVENT____META_CALL_PROGRESS
2/1 ConnConnectedEv for 2001
CallStateConnected-2001
2/1 CallCtlConnEstablishedEv for 2001
2/1 TermConnActiveEv for SEP003094C2A0B0
2/1 CallCtlTermConnTalkingEv for SEP003094C2A0B0
NEW META EVENT____META_CALL_PROGRESS
2001 が 2002 にコールする
NEW META EVENT____META_CALL_PROGRESS
2/1 CallCtlTermConnHeldEv for SEP003094C2A0B0
CallStateHold 2001
92479

参加 / 任意会議：ページ 2



コンサルト会議

コンサルト会議のメッセージ フローは、任意会議のフローと同じです。

拡張された回線をまたいで参加（Join Across Lines）

拡張された回線をまたいで参加（Join Across Lines）メッセージ フローを次の表に説明しています。A、C、D、E、および F は異なる端末のアドレスです。B1 と B2 は同じ端末 TermB のアドレスです。

操作	イベント
アプリケーションは GC1.conference(GC2) を呼び出して 2 つの電話会議をチェーンして、B1 および B2 で 2 つのコールの会議を開く。	**A、C、および B1 の CallObserver へのイベント：** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection は Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC1 を返す。
	B2、D、E での CallObserver のイベント： ConnDisconnectedEv B2 GC2 Cause=NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は Conference-1 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-1 と Conference-2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC1 と GC2 を返す。

操作	イベント
アプリケーションは GC2.conference(GC1) を呼び出して、2 つの電話会議をチェーンする。	**B2、D、E での CallObserver のイベント：** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は Conference-1 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-1 の Connection を返す 。 Ev.getConferenceChain().getChainedConferenceCalls() は GC2 を返す。
	A、B1、C での CallObserver のイベント： ConnDisconnectedEv B1 GC1 Cause=NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は Conference-2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC1 を返す。

操作	イベント
A、B1、C は conference-1（GC1）に参加し、B1、D、E は conference-2（GC2）に参加し、B2、F、G は conference-3（GC-3）に参加している。 アプリケーションは GC1.conference(GC2, GC3) を開始し、B1 をコントローラとして設定して、会議を開催する。	**A、B1、C での CallObserver のイベント：** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は Conference-2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC1 を返す。 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は Conference-3 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-2 と Conference-3 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC2 と GC3 を返す。

操作	イベント
	B1、D、E での CallObserver のイベント： ConnDisconnectedEv B1 GC2 Cause=NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は Conference-1 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-1-GC2 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC2 を返す。
	B2、F、G での CallObserver のイベント： ConnDisconnectedEv B2 GC3 Cause=NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC3 Ev.getAddedConnection は Conference-1 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は Conference-1 の Connection を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は GC3 を返す。

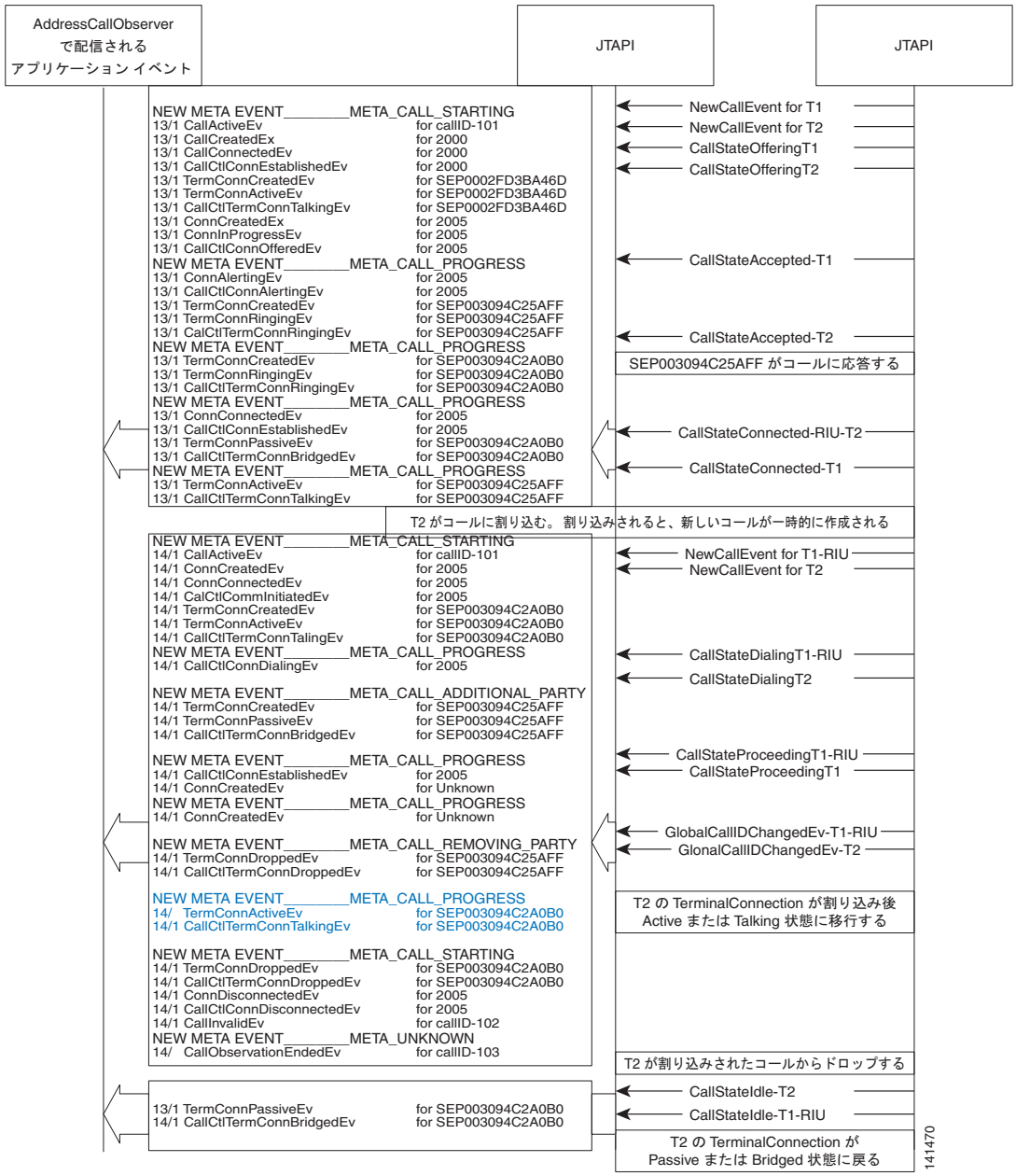
操作	イベント
アプリケーションはリクエスタを B2 と設定し、GC2.conference(GC1) を呼び出し、getControllerAddress() は B2 を返す。 getOriginalControllerAddress() は B1 を返す。	A CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
上の設定で、アプリケーションが要求コントローラとして B1 を使用する場合、getControllerAddress() は B1 を返す。 getOriginalControllerAddress() は B1 を返す。	イベントは上記と同じ

割り込みとプライバシー

割り込みとプライバシーに関するメッセージ フローを次に示します。

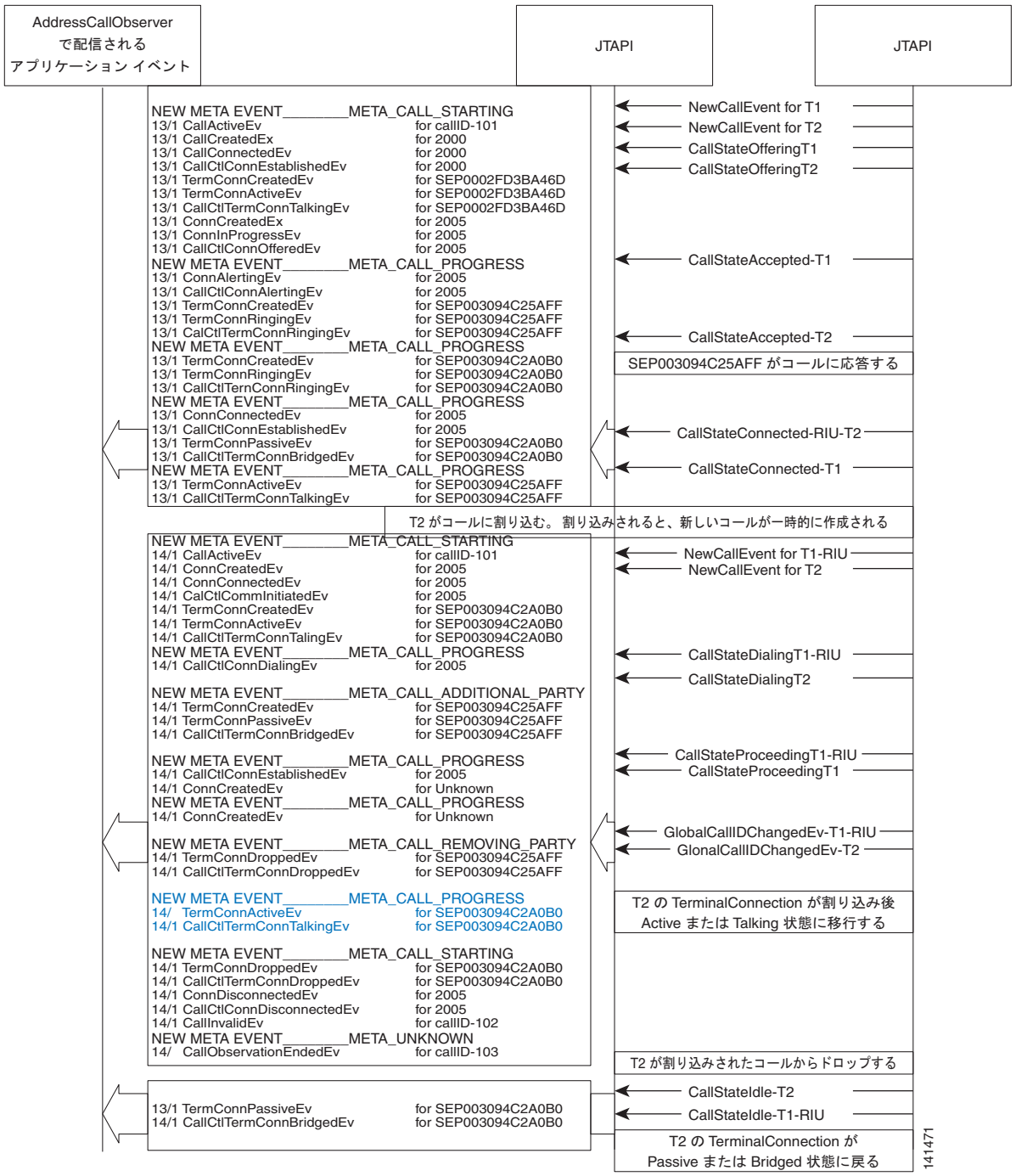
割り込み

シナリオ：2005 は端末 SEP003094C25AFF（T1）および端末 SEP00394C2A0B0（T2）上の共有回線である。 2000 が 2005 にコールを開始し、2005-T1がコールに応答する。 T2 がコールに割り込む

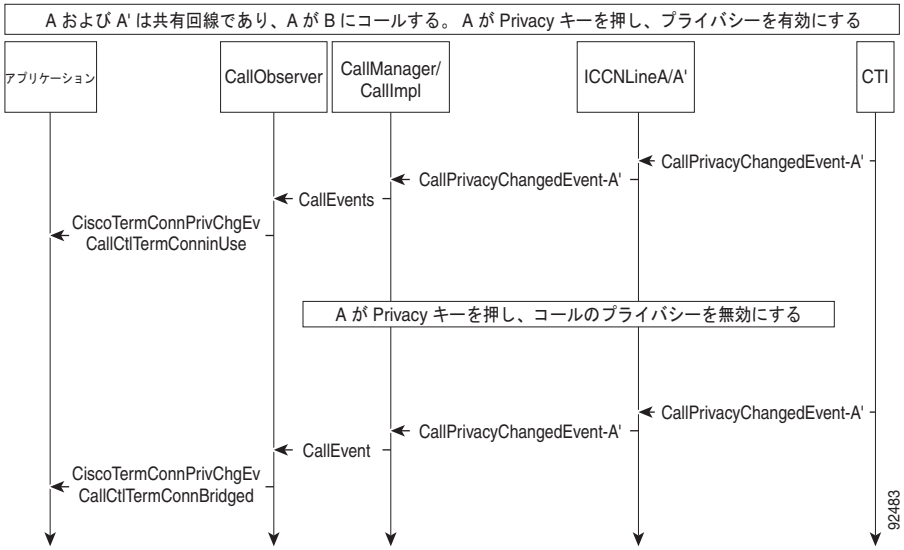


C 割込

シナリオ：2005 は端末 SEP003094C25AFF（T1）および端末 SEP00394C2A0B0（T2）上の共有回線である。 2000 が 2005 にコールを開始し、2005-T1がコールに応答する。 T2 がコールにC割り込みする

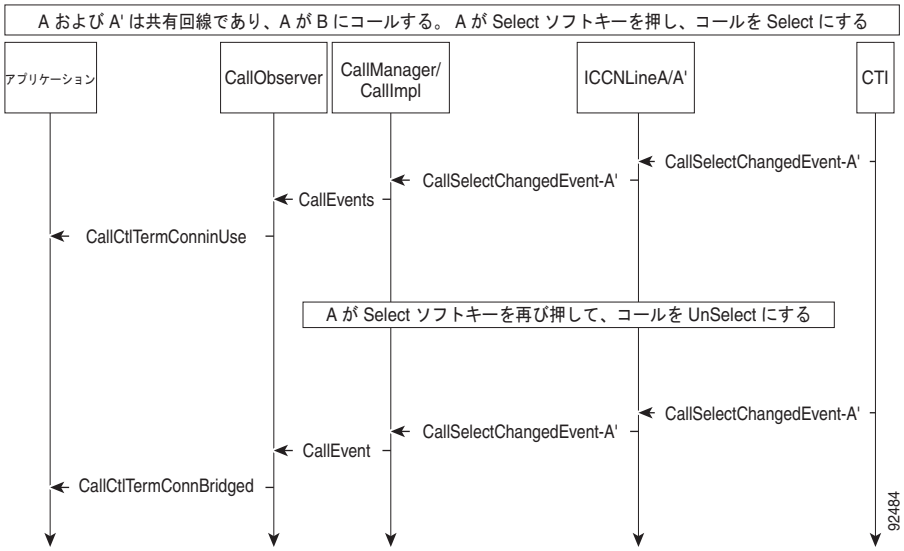


プライバシー



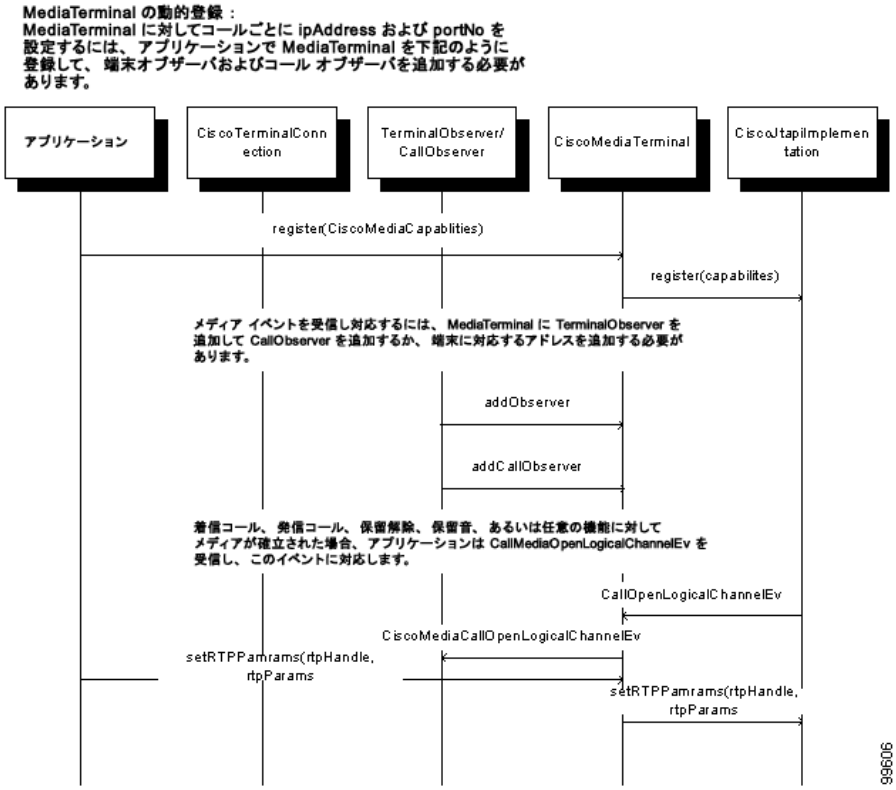
CallSelect と UnSelect

CallSelect と UnSelect に関するメッセージ フローを次に示します。



コールごとの CTIPort 動的登録

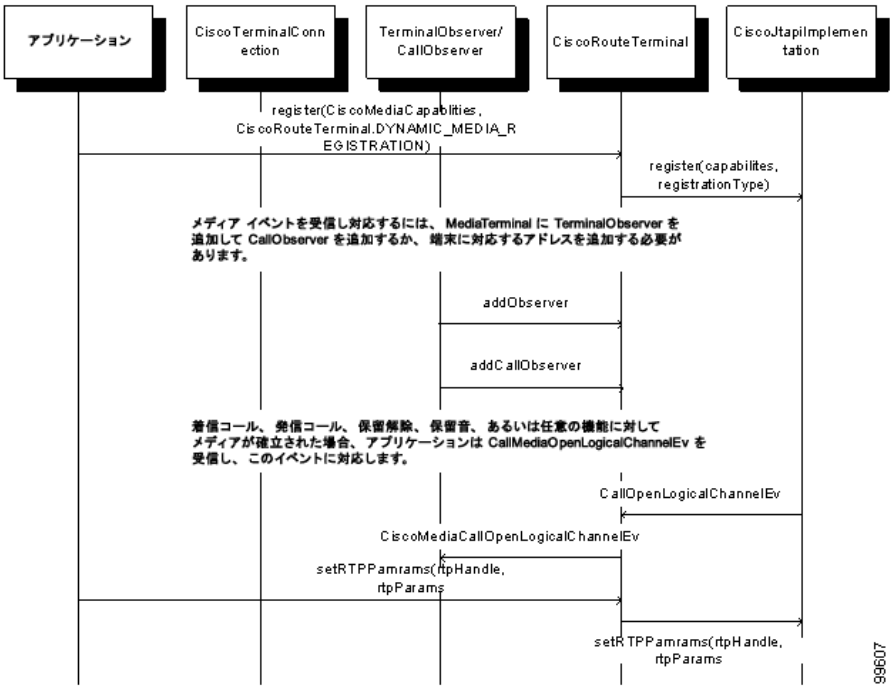
コールごとの CTIPort 動的登録に関するメッセージ フローを次に示します。



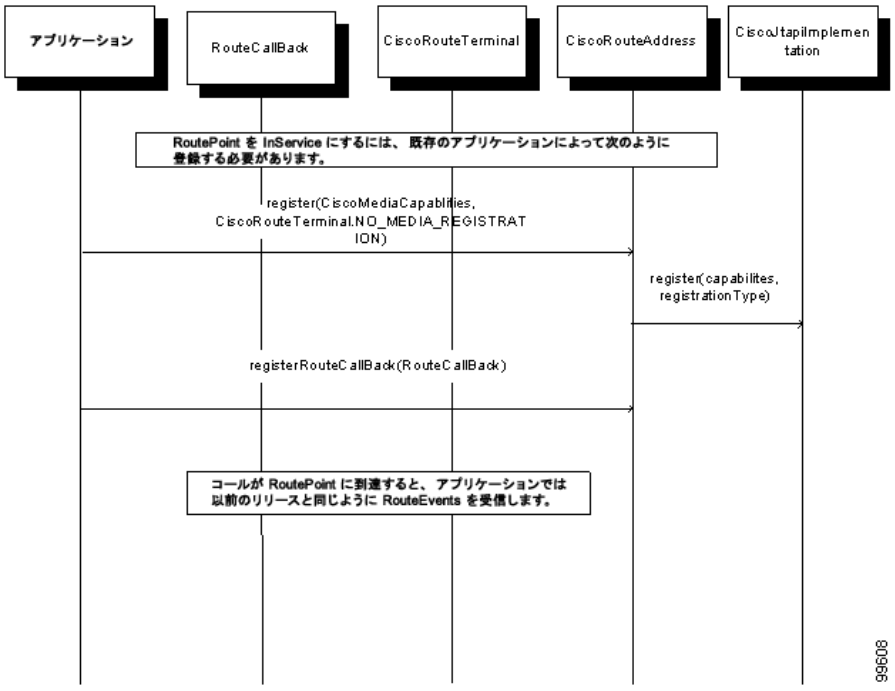
ルート ポイントでのメディア終端

ルート ポイントでのメディア終端に関するメッセージ フローを次に示します。

ルート ポイントでのメディア終端：
RoutePoint に対してコールごとに ipAddress および portNo を設定するには、アプリケーションで RouteTerminal を下記のように登録して、端末オブザーバおよびコール オブザーバを追加する必要があります。



ルート ポイントでのメディア終端：
ルーティングのためだけに RoutePoint を使用する既存のアプリケーションでは、次のように登録する必要があります。



リダイレクト時のオリジナルの着信者番号

次のシナリオは、リダイレクト時におけるオリジナルの着信者番号に関するメッセージ フローを示しています。

シナリオ 1

- A、B、C がアプリケーション コントロール リストにある。
- D がコントロール リストにない。
- A が B にコールする。
- B は、C を preferredOriginalCalledParty として D にコールをリダイレクトする。

アプリケーションには、A と B について次のイベントが示されます。

原因メタ イベント	コール	イベント	フィールド
META_CALL_ADDING_PARTY	Call 1	D の ConnCreatedEv D の ConnConnectedEv D の CallCtlConnEstablishedEv	CallingParty=A CalledParty = B **LastRedirectedParty=C** CurrentCalledParty=D
META_CALL_REMOVE_PARTY	Call 1	B の ConnDisconnectedEv B の CallCtlConnDisconnectedEv B の TermConnDroppedEv B の CallCtlTermConnDroppedEv B の CallObservationEndedEv	CallingParty=A CalledParty = B **LastRedirectedParty=C** CurrentCalledParty=D

（注） クラスタ内の通話者の場所、負荷、その他の条件によっては、指定されたイベント グループが同じ順序になく、変更される場合があります。

シナリオ 2

- A、B、C がコントロール リストにない。
- D がアプリケーション コントロール リストにある。
- A が B にコールする。
- B は、C を preferredOriginalCalledParty として D にコールをリダイレクトする。

アプリケーションには、D について次のイベントが示されます。

原因メタ イベント	コール	イベント	フィールド
META_CALL_STARTING	Call1	CallActiveEv D の ConnCreatedEv D の ConnInProgressEv D の CallCtlConnOfferedEv A の ConnCreatedEv A の CallCtlConnInitiatedEv	CallingParty=A CalledParty = D **LastRedirectedParty=C** CurrentCalledParty=D

原因メタ イベント	コール	イベント	フィールド
META_CALL_PROGRESS	Call1	D の ConnAlertingEv D の CallCtlConnAlertingEv D の TermConnCreatedEv D の CallCtlTermConnRingingEv A の ConnConnectedEv A の CallCtlConnEstablishedEv	CallingParty=A CalledParty = D **LastRedirectedParty=C** CurrentCalledParty=D
META_CALL_PROGRESS	Call	D の ConnConnectedEv D の CallCtlConnEstablishedEv D の TermConnActiveEv D の CallCtlTermConnTalkingEv	CallingParty=A CalledParty = D **LastRedirectedParty=C** CurrentCalledParty=D

シングル ステップ転送

アドレス A、B、および C がコントロール リストにあり、B を転送コントローラとして A と B の間のコールが C へ転送されます。アプリケーションには次のイベントが示されます。

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

操作	Address A（5001） Terminal CTIP1	Address B（5002） Terminal CTIP2	Address C（5003） Terminal CTIP3
Call.transfer(string) （続き）	CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 5003 CAUSE_NORMAL CallCtlConnEstablishedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL	NEW META EVENT_____META_UNKNOWN CallObservationEndedEv Cause: CAUSE_NORMAL	ConnAlertingEv 5003 Cause: CAUSE_NORMALCallCtlConnAlertingEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv CTIP3 TermConnRingingEv CTIP3Cause: CAUSE_NORMAL CallCtlTermConnRingingEvImpl CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStartedEv Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv Cause: CAUSE_NORMAL ConnConnectedEv 2004 Cause: CAUSE_NORMAL CallCtlConnEstablishedEv 5003Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

操作	Address A（5001） Terminal CTIP1	Address B（5002） Terminal CTIP2	Address C（5003） Terminal CTIP3
Call.transfer(string) （続き）			TermConnActiveEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPInputStoppedEv Cause: CAUSE_NORMAL CiscoRTPOutputStoppedEv Cause: CAUSE_NORMAL ConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5001 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallCtlTermConnDroppedEv CTIP3 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL ConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallCtlConnDisconnectedEv 5003 Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL META_UNKNOWN CallInvalidEv [#32] Cause: CAUSE_NORMAL

発信側番号の変更

次のシナリオは、発信側番号の変更に関するメッセージ フローを示しています。

シナリオ 1

アプリケーションが、デバイスの Route Point（RP）を制御し、RP を登録する。

A と B は PNO で、Cisco Unified Communications Manager クラスタ内にある。

A が RP にコールする。

コールが RP に到達する。

操作	イベント	フィールド
コールが RP に到達する。	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA （A に関連付けられた端末）
アプリケーションが次のように呼び出す。 selectRoute(routeselected[], callingsearchspace, modifiyingcallingnumber[]) where routeSelected[] = C callingSearchSpace = CiscoRouteSession.DEFAULT_SEARCH_SPACE	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA （A に関連付けられた端末） getRouteUsed () = C
アプリケーションが次のように呼び出す。 endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_END getRouteAddress () = RP

シナリオ 2

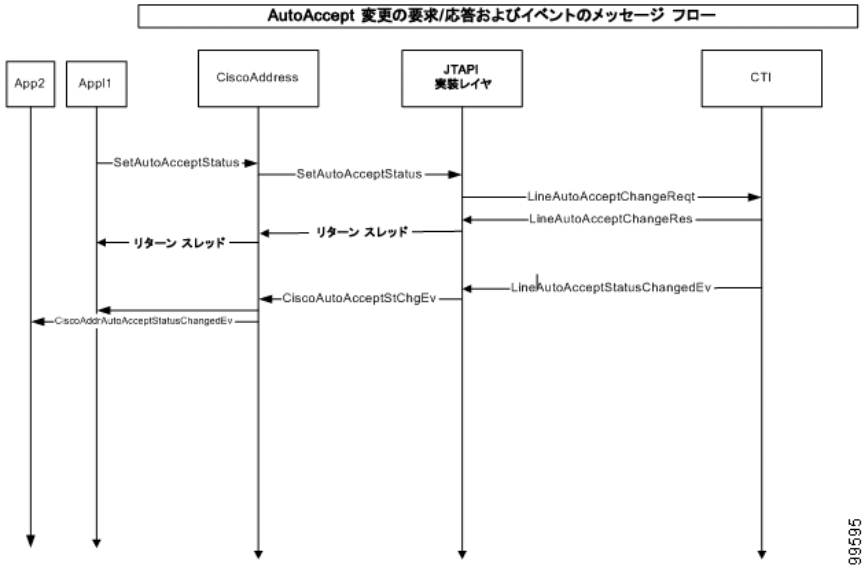
アプリケーションが A と B を制御している。

A から RP へのコールにより、発信側番号が M に変更されたコールが B へ selectRoute 処理される。

操作	イベント	フィールド
A が RP にコールするが、RP がコントロール リストにない。	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL	getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= getModifiedCallingAddress()=A getModifiedCalledAddress() =
	NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A	getCallingAddress() = A getCalledAddress() = getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= getModifiedCallingAddress()=A getModifiedCalledAddress() =
	NEW META EVENT_____META_CALL_PROGRESS CallCtlConnEstablishedEv A ConnCreatedEv RP ConnInProgressEv RP CallCtlConnOfferedEv RP	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()=A getModifiedCalledAddress() =B

操作	イベント	フィールド
この RP を制御している別のアプリケーションが、発信側番号を M に変更して B への selectRoute 処理を行う。	NEW META EVENT_____META_CALL_ ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B ConnDisconnectedEv RP CallCtlConnDisconnectedEv RP	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= **M** getModifiedCalledAddress() =B
	NEW META EVENT_____META_CALL_ PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= **M** getModifiedCalledAddress() =B
B が、コールに応答する。	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B	getCallingAddress() = A getCalledAddress() = B getLastRedirectedAddress ()= RP getCurrentCallingAddress ()= A getCurrentCalledAddress()= B getModifiedCallingAddress()= **M** getModifiedCalledAddress() =B

CTIPort および RoutePoint での AutoAccept



Forced Authorization Code と Customer Matter Code

シナリオ 1

アプリケーションが A と B を制御していて、B にコールを発信するには Forced Authorization Code（FAC）が必要です。

操作	イベント
A が call.Connect() を使用して B にコールする、または A が Call.Consult() を使用して B にコンサルト コールを発信する。	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv.FAC_REQUIRED

操作	イベント
アプリケーションが CiscoConnection.addToAddress を使用して追加の番号を入力する。	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B ConnConnectedEv B CallCtlConnEstablishedEv B
B が、コールに応答する。	TermConnActiveEv B

シナリオ 2

アプリケーションが A と B を制御していて、B にコールを発信するには FAC および CMC の両方が必要です。

操作	イベント
A が call.Connect() を使用して B にコールする、または A が Call.Consult() を使用して B にコンサルト コールを発信する。	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv.FAC_CMC_REQUIRED
アプリケーションが T302 タイマーの時間内に、CiscoConnection.addToAddress を使用して FAC コード番号を入力し、末尾に「#」を入力する。	NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv.CMC_REQUIRED

操作	イベント
アプリケーションが T302 タイマーの時間内に、CiscoConnection.addToAddress を使用して CMC コード番号を入力し、末尾に「#」を入力する。	NEW META EVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv B TermConnRingingEv B CallCtlTermConnRingingEv B
B が、コールに応答する。	ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv B CallCtlTermConnTalkingEv B

シナリオ 3

アプリケーションが A と B を制御しています。

B は CMC を必要としており、アプリケーションが無効なコードを入力します。

操作	イベント
A が call.Connect() を使用して B にコールする、または A が Call.Consult() を使用して B にコンサルト コールを発信する。	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW META EVENT_____META_CALL_PROGRESS CiscoToneChangedEv ToneType = CiscoTone.ZIPZIP cause = CiscoCallEv.CAUSE_FAC_CMC getWhichCodRequired = CiscoToneChangedEv.CMC_REQUIRED

操作	イベント
アプリケーションが T302 タイマーの制限時間内に、CiscoConnection.addToAddress を使用して不正な CMC コード番号を入力し、末尾に「#」を入力する。 アプリケーションがリオーダー音を受信する。	NEW META EVENT_____META_CALL_PROGRESS ConnFailedEv A CallCtlConnFailedEv A getCiscoCause () = CiscoCallEv.FAC_CMC NEW META EVENT_____META_CALL_ENDING TermConnDroppedEv CallCtlTermConnDropped ConnDisconnectedEv CallCtlConnDisconnectedEv CallInvalidEv CallObservationEndedEv

シナリオ 4

アプリケーションが A と B の両方を制御していて、A が B にコールし、B が、FAC および CMC の両方を必要とする C にコールをリダイレクトします。

操作	イベント
A が call.Connect() を使用して B にコールする、または A が Call.Consult() を使用して B にコンサルト コールを発信する。	NEW META EVENT_____META_CALL_STARTING CallActiveEv Cause: CAUSE_NEW_CALL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL TermConnCreatedEv SEPA Cause: Other: 0 TermConnActiveEv SEPA Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv SEPA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL NEW META EVENT_____META_CALL_PROGRESS CallCtlConnDialingEv A NEW METAEVENT_____META_CALL_ADDITIONAL_PARTY ConnCreatedEv B ConnInProgressEv B CallCtlConnOfferedEv B NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv B CallCtlConnAlertingEv B TermConnCreatedEv SEPB TermConnRingingEv SEPB CallCtlTermConnRingingEv SEPB ConnConnectedEv B CallCtlConnEstablishedEv B TermConnActiveEv SEPB CallCtlTermConnTalkingEv SEPB

操作	イベント
B が C へのリダイレクト要求を発行し、FAC および CMC コードを渡す。	NEW META EVENT_____META_CALL_REMOVING_PARTY TermConnDroppedEv SEPB CallCtlTermConnDroppedEv SEPB Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnDisconnectedEv B Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnInProgressEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnAlertingEv A Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED NEW META EVENT_____META_CALL_PROGRESS ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoCause: CAUSE_NOERROR

シナリオ 5

アプリケーションが、デバイスの Route Point（RP）を制御し、RP を登録します。

A と B は PNO で、Cisco Unified Communications Manager クラスタ内にあります。

操作	イベント	フィールド
コールが RP に到達する。	RouteEvent	State = ROUTE getCurrentRouteAddress () = RP getCallingAddress () = A getCallingTerminal () = SEPA（A に関連付けられた端末）

操作	イベント	フィールド
アプリケーションが次のように呼び出す。 selectRoute(routeselected[], callingsearchspace, modifiyingcallingnumber[], preferredOriginalCdNumber[], preferredOriginalCdOption[], facCode[], cmcCode[]) where routeSelected[] = B callingSearchSpace = CiscoRouteSession.DEFAULT_SEARCH_SPACE modifyingCgNumber = null, preferredOriginalCdNumber = null, preferredOriginalCdOption = CiscoRouteSession.DONOT_RESET_ORIGINALCALLED, facCode[] = "facCode for B" cmcCode[] = "cmcCode for B"	RouteUsedEvent	State = ROUTE_USED getCallingAddress () = A getCallingTerminal () = SEPA（A に関連付けられた端末） getRouteUsed () = B
アプリケーションが次のように呼び出す。 endRoute (ERROR_NONE)	RouteEndEvent	State = ROUTE_END getRouteAddress () = RP

スーパー プロバイダーのメッセージ フロー

アプリケーションが、アドレス 2000 および 2001 を持つ CTIPort1 の端末の作成を試みます。次のイベントがアプリケーションに送信されます。

番号	操作	イベント
1	アプリケーションは CiscoProvider.CreateTerminal(CTIPort1) を呼び出す。ここで CiscoProviderCapabilities. canObserveAnyTerminal() は TRUE を返す。	JTAPI が CiscoTerminal オブジェクトを返し、次のイベントが送信される。 CiscoTermCreatedEv CTIPort1<---------- CiscoAddrCreated 2000<---------- CiscoAddrCreated 2001<----------
2	アドレス 2001 が存在する端末をアプリケーションがすでに持っている場合、つまり 2001 が SharedLine アドレスである場合。 アプリケーションは CiscoProvider.CreateTerminal(CTIPort1) を呼び出す。	JTAPI が CiscoTerminal オブジェクトを返し、次のイベントが送信される。 CiscoTermCreatedEv CTIPort1<---------- CiscoAddrCreated 2000<----------11 CiscoAddrAddedToTerminalEv 2001<--------

番号	操作	イベント
3	アプリケーションが次のように呼び出す。 CiscoProvider. CreateTerminal(CTIPortX) を呼び出す。 ここで CTIPortX は Cisco Unified Communications Manager クラスタに存在しない。	JTAPI が例外 InvalidArgumentException をスローする。
4	アプリケーションが次のように呼び出す。 CiscoProvider. CreateTerminal(CTIPort1) を呼び出す。 ここで CiscoProviderCapabilities.canObserve AnyTerminal() は FALSE を返す。	JTAPI が例外 PrivilegeViolationException をスローする。

スーパープロバイダーと変更通知の拡張の使用例

新しいフェールオーバー シナリオと変更通知を処理するためにアプリケーションに送信される新しいイベントが JTAPI に追加されました。これによりフェールオーバー シナリオの処理や、スーパープロバイダー モードと通常ユーザ モードの切り替えにかかる時間の処理について JTAPI が拡張されます。

シナリオ 1

スーパープロバイダー ユーザが、プロバイダーをオープンして、コントロール リストにないいくつかのデバイスをスーパープロバイダー モードでオープンします。admin ページから Superprovider 特権を解除します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。JTAPI から、オープンまたは取得したデバイスのうちコントロール リストにないものすべてについて、CiscoTermRemovedEv が送信されます。JTAPI はアプリケーションにプロバイダー OOS を送信し、コントロール リストに含まれないデバイスに CiscoTermRemovedEv を送信して、CTI への接続を再オープンします。接続が成功すると、JTAPI はプロバイダーのイン サービス イベントをアプリケーションに送信します。そうでない場合は、プロバイダーをクローズします。

シナリオ 2

通常ユーザが、プロバイダーをオープンして、コントロール リストにあるいくつかのデバイスをオープンします。admin ページから Superprovider 特権をユーザに追加します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。ユーザはコントロール リストにないデバイスを取得またはオープンできるようになります。

シナリオ 3

通常ユーザが、プロバイダーをオープンして、いくつかのパーク DN をオープンします。admin ページからユーザのパーク DN のモニタ特権を解除します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。JTAPI はすべてのパーク DN アドレスをクリーンアップします。

シナリオ 4

通常ユーザがプロバイダーをオープンします。admin ページからユーザにパーク DN のモニタ特権を追加します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。アプリケーションはパーク DN のモニタリング機能を登録し、パーク DN のモニタリングを実行できるようになります。

シナリオ 5

通常ユーザがプロバイダーをオープンします。admin ページからユーザの「modify calling party」特権を解除します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。アプリケーションは、リダイレクト中に発番号を変更できません。これを実行しようとすると、JTAPI がエラーをスローします。

シナリオ 6

通常ユーザがプロバイダーをオープンします。admin ページからユーザに「modify calling party」特権を追加します。

アプリケーションは CiscoProviderCapabilityChangedEvent イベントを受け取ります。アプリケーションは、リダイレクト中に発番号を変更できます。

シナリオ 7

スーパープロバイダー ユーザが、プロバイダーをオープンして、コントロール リストにないデバイスを取得します。admin ページからデバイスを削除します。

アプリケーションは CiscoTermRemovedEv イベントを受け取ります。デバイスは JTAPI の観点からクローズされます。

QSIG パス置換

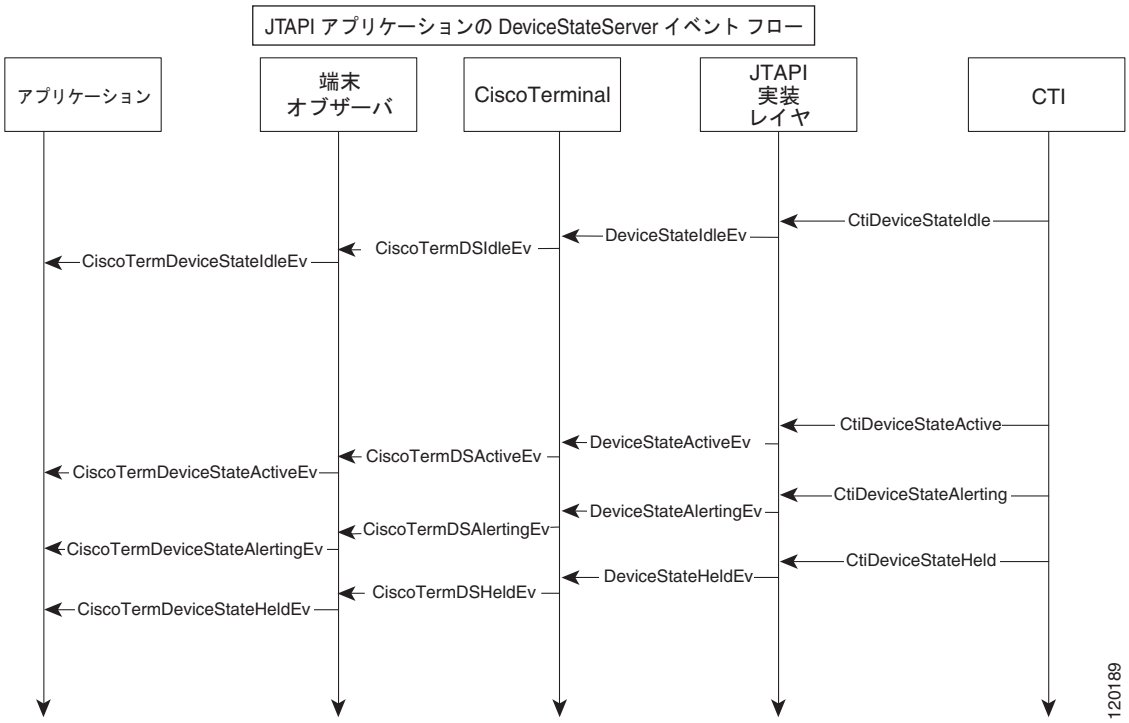
次の表は、Q.Signaling（QSIG）トランクで接続された PBX 間のコールが転送されたときに、アプリケーションに送信される JTAPI イベントを示しています。また、QSIG パス置換機能によってリアルタイム パス（RTP）が最適化されたときにアプリケーションに送信されるイベントも示しています。

トランスレーション パターンによってパターンが変更されている場合、QSIG トランクを介するコールには遠端との接続がないことがあります。つまり、アプリケーションでトロンボーン ケースのコールが 2 つ確認されているときに、B がそれらのコールの共通の接続として機能するとは限りません。

番号	操作	イベント
1	A は CM1、B は CM2、C は CM3 にそれぞれ登録されている。 A が B にコールし（GC1）、B がそのコールを C に転送する。アプリケーションは C を監視する。コールが C に接続されると、PR 機能により、パスが置換される。 この動作は、B でコール転送が行われるシナリオでも同じ （着信者がコールを転送）。	次のイベントがアプリケーションに送信される。 CallCtrlConnectionEstablishedEv A CallCtrlConnectionDisConnectEv B OpenLogicalChannelEvent : C が CTI デバイス（CTIPort および RP に動的に登録される）の場合
2	A は CM1、B は CM2、C は CM3 にそれぞれ登録されている。B が C にコールし、C が応答し、B がコールを A に転送する。A が応答する。アプリケーションは C だけを監視している （発信者がコールを転送）。	この場合、転送が成立すると、A と C の両方が着信者になる。コールに対して応答があると、PR によりパスが置換される。この場合、A と C は IP フォンであり、PR 機能の動作の一環としてディスプレイが更新される。これにより、A または C が発信者になる。 JTAPI イベント： GC1: CallCtlConnEstablishedEv A GC1: CallCtlConnDisconnectedEv B
3	3.トロンボーンの場合：A は CM1、B は CM2、C は CM1 にそれぞれ登録されている。A が B（GC1）にコールし、B が応答してコールを C（GC2）に転送する。パス置換により、CM2 がバイパスされて A と C が接続される。アプリケーションは A と C の両方を監視している （着信者がコールを転送）。	GC1 コール オブザーバ： GC1: CallCtlConnEstablishedEv C GC1: CallCtlConnDisconnected B PR 機能によってパスが置換される前に、アプリケーションが 2 つのコールを確認する。1 つは A および C（外部）に接続されている GC1 で、もう 1 つは C および A（外部）に接続されている GC2。 PR 機能によりパスがパスが置換されると、GC1 が GC2 に変更されるか、GC2 が GC1 に変更される。 A の GCID が、GC1 から GC2 に変更された場合 GC1: CiscoCallChangedEv (oldGCID=GC1,newGCID=GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: TermConnTalkingEvent for TerminalA cause= CAUSE_QSIG_PR

番号	操作	イベント
4	トロンボーンの場合：A は CM1、B は CM2、C は CM1 にそれぞれ登録されている。B が A にコールし、B がコールを C に転送する。パス置換により、CM2 がバイパスされて A と C が接続される。アプリケーションは A と C の両方を監視している（発信者がコールを転送）。	PR 機能によってパスが置換される前に、アプリケーションが 2 つのコールを確認する。1 つは A および B（外部）に接続されている GC1 で、もう 1 つは C および B（外部）に接続されている GC2。この場合、アプリケーションは転送開始イベントを受信しない。 PR 機能によってパスが置換されると、A および C のディスプレイが更新され、パスが置換されて、GCID が変更される。A の GCID が変更されて発信者になった場合、次の JTAPI イベントが発生する。 GC1: CiscoCallChangedEv (GC1 to GC2) GC1: CallCtlConnDisconnected for A GC1: CallCtlConnDisconnected for C GC1: CallInValid GC2: ConnCreatedEv A GC2: ConnConnectedEv A GC2: TermConnTalkingEvent for TerminalA cause= CAUSE_QSIG_PR
5	A は CM1、B は CM2、C は CM1 にそれぞれ登録されている。A が B にコールし、B がそのコールを C に転送する。C が応答し、パス置換が完了する前に、C がパークやリダイレクトなどの機能を呼び出す。	パス置換が放棄される。
6	状況によっては、機能（リダイレクト、パーク、転送など）の要求が無視される。この状況は、セットアップ要求が送出され、ローカルの CM が相手側からの接続を待っている場合に発生する。	JTAPI 機能要求に関する例外が JTAPI からスローされる。
7	PR 機能が RTP パスの切り替えを試みたときに CM がダウンしていた場合、アプリケーションが無音状態を受信することがある。これは、すでに接続されているコールに対して可能性がある。	イベントなし JTAPI アプリケーション：コールを終了する

デバイス ステート サーバ



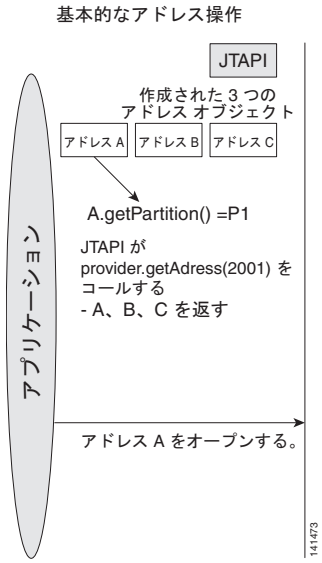
パーティションのサポート

JTAPI のアドレスのハッシュ機構が変更されたため、この機能によるアドレス検索やロード テスト時のパフォーマンスは低下することが予想されます。

getPartition() API の使用

次の例は、JTAPI およびアプリケーションで getPartition() を使用して DN が同じでパーティションが異なるアドレスを区別する方法を示したものです。

例 A-1　　getPartition() API の使用



ここでは、3 つの異なるパーティションに属する A（2001, P1）、B（2001, P2）、および C（2001, P3）という 3 つのアドレスがあるとします。2001 は DN を表し、P1、P2、および P3 はそれぞれのパーティションを表します。

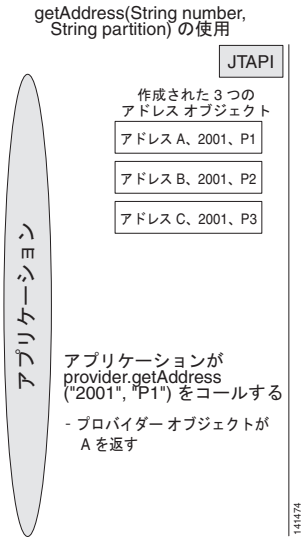
JTAPI が provider.getAddress("2001") をコールすると、プロバイダー オブジェクトは A、B、および C の 3 つのアドレス オブジェクトを含む配列を返します。これはこの 3 つがすべて同じ DN を持っているためです。

アプリケーションおよび JTAPI は、アドレス オブジェクトの getPartition() メソッドを使用して 3 つのアドレスを区別します。

getAddress(String number, String partition) の使用

次の例は、DN が同じでパーティションが異なる複数のアドレスがある場合に、JTAPI が getAddress(String number, String partition) API を使用して特定の DN およびパーティションに対応するアドレス オブジェクトを取得する方法を示しています。

例 A-2　getAddress(String number, String partition) の使用



この例では、3 つの異なるパーティションに属する 3 つのアドレスがあります。それぞれ A（2001, P1）、B（2001, P2）、および C（2001, P3）と表します。2001 は DN を表し、P1、P2、P3 はそれぞれのパーティションを表します。

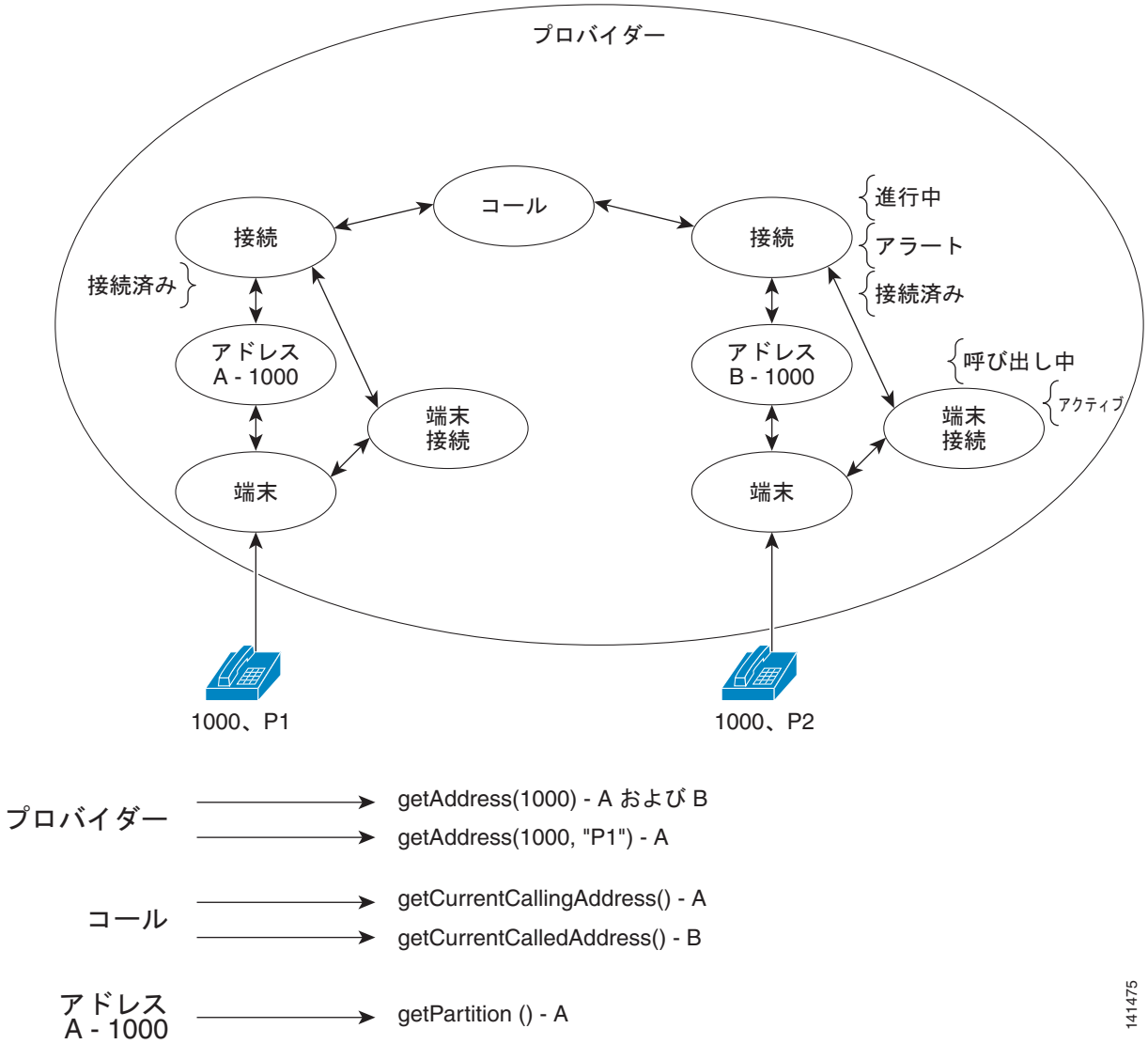
JTAPI が ***provider.getAddress("2001", "P1")*** をコールすると、プロバイダー オブジェクトは API で指定したのと同じ DN（2001）とパーティション情報（P1）を持つアドレス オブジェクトを返します。この場合、アプリケーションにはアドレス オブジェクト A が返されます。

単純なコールのシナリオ

A が B にコールするシナリオを考えます。A は DN が 1000 であり、同じく DN が 1000 である B にコールします。A はパーティション P1 に属し、B はパーティション P2 に属します。次の図は、このシナリオにおけるさまざまなイベントおよび API コールの結果を示しています。これはパーティション サポート機能に関連するものです。

例 A-3　　単純なコール シナリオ

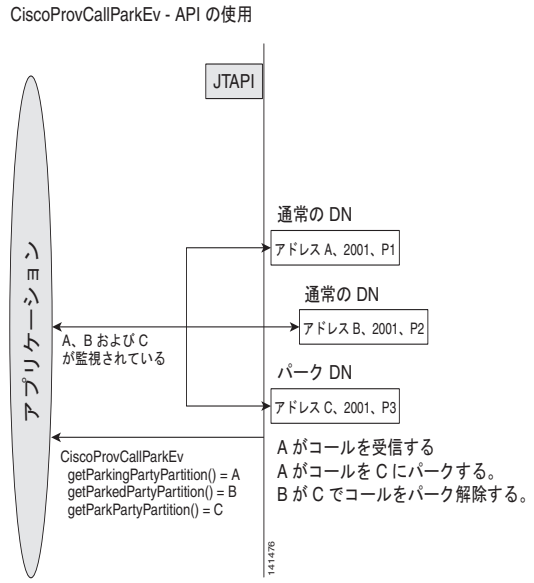
シナリオ：回線 x1000 :P1" から 回線 x1000 "P2" へのコール



パーク DN

JTAPI ではパーク DN もアドレスとして扱われます。そのため、パーク DN も通常の DN と同様に処理されます。次のメッセージ フローは、パーク DN が使用されているコールでアプリケーションがパーク DN のパーティション情報を使用する方法を示したものです。

例 A-4　　パーク DN シナリオ



アプリケーションがパーク DN をモニタリングする際、パーク DN は通常の DN と同じものにできます（それぞれが異なるパーティションに属している場合）。

このケースでは、C は A および B と同じ DN 値を持つパーク DN であり、異なるパーティションに属しています。

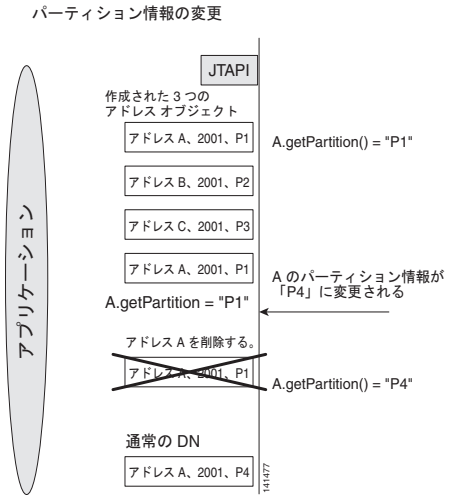
A がコールを受信してそれを C にパークします。B がそのコールをパーク解除します。コールがパークおよびパーク解除されると、CiscoProvCallParkEv が生成されます。API

getParkingPartyPartition()、getParkedPartyPartition()、および getParkPartyPartition() は、図に示したような関連するアドレス オブジェクトを返します。

パーティションの変更

パーティションのアトリビュートは、アドレスの DN アトリビュートと同様のものです。そのためパーティションのアトリビュートを変更するたび、アドレス オブジェクトを破棄して作成し直す必要があります。アドレスのパーティション情報を変更すると、JTAPI が再起動され、その際に現在のアドレス オブジェクトが削除されて、パーティション情報の変更を反映した新しいアドレス オブジェクトが作成されます。

例 A-5　　パーティションの変更



アドレスのパーティション情報を変更すると、アドレス オブジェクトは破棄されて、新しいアドレス オブジェクトが作成されます。

新しいアドレス オブジェクトには新しいパーティション情報が格納されます。

上の例では、アドレス A のパーティション文字列が P4 に変更されています。そのため A の現在のアドレス オブジェクトは削除されて、新しいアドレス オブジェクトが作成されます。

A.getPartition() を使用して古いアドレス オブジェクトを問い合せると「P1」が返され、同じ問い合せを新しいオブジェクトに対して行うと「P4」が返されます。

アドレスのパーティションが変更されたら、アプリケーションはアドレス オブジェクトの問い合せを行ってパーティション情報を更新する必要があります。

JTAPI のパーティションのサポート

次のすべての使用例では、JTAPI がオープンするすべての回線のパーティション情報を CTI が応答メッセージで提供し、JTAPI は保持する全アドレスのパーティション情報を保管していることを想定しています。

番号	事前条件	シナリオ	予想される動作	結果
1	A と B は同じクラスタ内にある 2 つのアドレスであり、DN は同じでパーティションが異なる（P1、P2）。A が B にコールする。A の CSS には、最初に B のパーティションが含まれる。	異なるデバイス上の同じ DN でパーティションが異なるアドレス間でのコールは通常成功する。	A および B にそれぞれ 1 つずつ、計 2 つのアドレス オブジェクトが作成される。該当するすべてのコール関連イベントが、両方のアドレスに配信される。	A と B の間でコールが確立される。
2	A と B は同じデバイスにある 2 つのアドレスで、DN は同じでパーティションが異なり（P1、P2）、同じクラスタに含まれる。A が B にコールする。A の CSS には、最初に B のパーティションが含まれる。	同じデバイス上の同じ DN でパーティションが異なるアドレス間でのコールは通常成功する。	A および B にそれぞれ 1 つずつ、計 2 つのアドレス オブジェクトが作成される。該当するすべてのコール関連イベントが、両方のアドレスに配信される。	A と B の間でコールが確立される。
3	A、B、および C はそれぞれ異なる DN を持つ 3 つのアドレス （P1、P2、P3）。パーク DN は、DN が C と同じでパーティションが異なる（P4）。	A が B にコールする。B はコールをパークする。C は C の DN と同じパーク DN からのコールをパーク解除する。	JTAPI は C がパーク DN からのコールをパーク解除することを認める。	C がパーク DN からのコールのパーク解除に成功する。
4	A、B、および C は、DN が同じでパーティションが異なる（P1、P2、P3）3 つのアドレス。パーク DN は DN が同じでパーティションが異なる（P4）。	A が B にコールし、B がパーク DN でコールをパークする。C がコールをパーク解除する。	JTAPI は C がパーク DN からのコールをパーク解除することを認める。	A は B にコールできる。B は通常、パーク DN でコールをパークできる。 C は通常、コールをピックアップできる。
5	A、B、および C は、DN が同じでパーティションが異なる（P1、P2、P3）3 つのアドレス。	JTAPI が getPartitionAddress（A の DN）をコールする。	A、B、および C に対応する 3 つのアドレス オブジェクトが返される。	JTAPI はパーティション情報および DN に基づいてアドレス オブジェクトを保持する。
6	A と B は、DN は同じでパーティションが異なる（P1、P2）アドレス。	アプリケーションが A および B のアドレス オブジェクトで getPartition() をコールする。		アドレスのパーティション文字列が正しく返される。

番号	事前条件	シナリオ	予想される動作	結果
7	A と B は、パーティションが異なる 2 つのアドレス（DN も異なる）。A および B は同じユーザのコントロール リストに含まれる。プロバイダーのオープンが完了し、回線がオープンされる。	JTAPI は DN が異なる場合に回線をオープンするための古い API をサポートしている。動作に変化はなし。	回線 A および B がオープンされる。それぞれ DN が異なるため、ユーザがパーティション情報を指定する必要はない。回線をオープンするには DN だけで十分だが、ユーザがパーティション情報を指定することもできる。これにより、両モードでの回線オープンが JTAPI でサポートされる。	A および B のアドレス オブジェクトが正常に作成される。
8	A はユーザのコントロール リストに含まれるアドレスで、ユーザがこれをオープンする。CM admin ページから、A のパーティション情報を変更する。この後、デバイスが再起動される。	DN のパーティション情報が変更される。JTAPI が新しいパーティション情報を反映する。	JTAPI は A の現在のアドレス オブジェクトを削除し、A が再度イン サービスになると新しいアドレス オブジェクトを作成する。このアドレス オブジェクトのパーティション情報を問い合せると、変更後のパーティション情報が返される。	新しいアドレス オブジェクトに新しいパーティション情報が反映される。

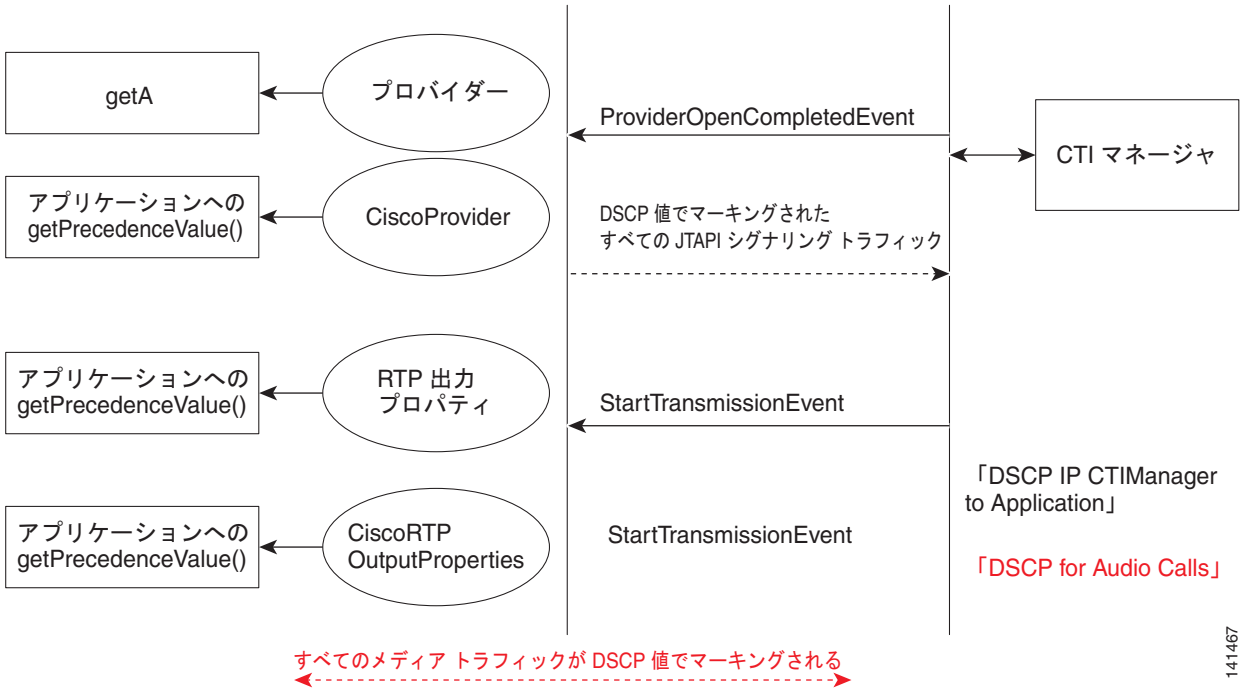
ヘアピン サポート

番号	事前条件	使用例	予想される動作	結果
1	IP フォン A および C は同じクラスタにあり、IP フォン B は別のクラスタにある。JTAPI は A および C を監視している。ゲートウェイは新しい通話者の情報をそれぞれの通話者に渡さない。転送コントローラは制御対象デバイスではないため、転送開始および終了イベントは発生しない。	A がゲートウェイを介して B にコールする。B がゲートウェイを介して C にコールを転送する。B が転送を実行し、シナリオから離脱する。これで IP フォン A および C が接続される。	A : B に接続されている。 A のタイプは CiscoAddress.Internal。 B のタイプは CiscoAddress.External。 C : B に接続されている。 C のタイプは CiscoAddress.Internal。 B のタイプは CiscoAddress.External。	A と C が接続される。
2	IP フォン A および C は同じクラスタにあり、IP フォン B は別のクラスタにある。JTAPI は A および C を監視している。ゲートウェイは新しい通話者の情報をそれぞれの通話者に渡すことができる。	A がゲートウェイを介して B にコールする。B がゲートウェイを介して C にコールを転送する。B が転送を実行し、シナリオから離脱する。これで IP フォン A および C が接続される。	A : C に接続されている。 A のタイプは CiscoAddress.Internal。 C のタイプは CiscoAddress.External。 C : A に接続されている。 C のタイプは CiscoAddress.Internal。 A のタイプは CiscoAddress.External。	A と C が接続される。

番号	事前条件	使用例	予想される動作	結果
3	IP フォン A および B は同じクラスタにあり、IP フォン C は別のクラスタにある。JTAPI は A および B を監視している。	A が B にコールする。B がゲートウェイを介して C に電話会議のコールを行う。B が会議を開催し、A、B、および C のすべてが会議に参加する。	A および B では、ConferenceCallStateChanged イベントに次のタイプの participantInfo がある。 A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A、B、および C が電話会議に接続される。
4	IP フォン A および B は同じクラスタにあり、IP フォン C は別のクラスタにある。JTAPI は A、B、および C を監視している。	A が B にコールする。B がゲートウェイを介して C に電話会議のコールを行う。B が会議を開催し、A、B、および C のすべてが会議に参加する。	A および B では、ConferenceCallStateChanged イベントに次のタイプの participantInfo がある。 A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A、B、および C が電話会議に接続される。
5	IP フォン A、B、および C は同じクラスタにあり、IP フォン D は別のクラスタにある。JTAPI は A、B、および C を監視している。ゲートウェイは新しい通話者の情報をそれぞれの通話者に渡すことができる。	A が B にコールする。B がゲートウェイを介して D に電話会議のコールを行う。D はそのコールを C に転送する。B が会議を開催し、A、B、および C のすべてが会議に参加する。	A および B では、ConferenceCallStateChanged イベントに次のタイプの participantInfo がある。 A: CiscoAddress.Internal B: CiscoAddress.Internal C: CiscoAddress.External	A、B、および C が電話会議に接続される。
6	IP フォン A、B、および C は同じクラスタにあり、IP フォン D は別のクラスタにある。JTAPI は A、B、および C を監視している。ゲートウェイは新しい通話者の情報をそれぞれの通話者に渡さない。	A が B にコールする。B がゲートウェイを介して D に電話会議のコールを行う。D はそのコールを C に転送する。B が会議を開催し、A、B、および C のすべてが会議に参加する。	A および B では、ConferenceCallStateChanged イベントに次のタイプの participantInfo がある。 A: CiscoAddress.Internal B: CiscoAddress.Internal D: CiscoAddress.External	A、B、および C が電話会議に接続される。
7	IP フォン A および C は同じクラスタにあり、IP フォン B は別のクラスタにある。JTAPI は A および C を監視している。	A がゲートウェイを介して B にコールする。B はコールを C にリダイレクトする。これで IP フォン A および C が接続される。	A：C に接続されている。 A のタイプは CiscoAddress.Internal。 C のタイプは CiscoAddress.External。 C：A に接続されている。 C のタイプは CiscoAddress.Internal。 A のタイプは CiscoAddress.External。	A と C が接続される。

QoS のサポート

図 A-1　QoS のサポートのコール フロー図



JTAPI QoS

QoS を Windows で機能させるには、次の手順を完了する必要があります。

ステップ 1　レジストリ エディタ（Regedt32.exe）を起動します。

ステップ 2　次のキーを探します。

HKEY_LOCAL_ MACHINE¥System¥CurrentControlSet¥Services¥Tcpip¥Parameters¥

（注）　レジストリ キーは 1 つのパスです。

ステップ 3　[編集] メニューで、[値の追加] をクリックします。

ステップ 4　DisableUserTOSSetting と入力します。

ステップ 5　[データ型] ボックスの [REG_DWORD] をクリックします。

ステップ 6　[OK] をクリックします。

ステップ 7　プロンプト ボックスに 0 と入力します。

ステップ 8　レジストリ エディタを終了します。

ステップ 9　コンピュータを再起動します。

詳細については、http://support.microsoft.com/default.aspx?scid=kb;ja-jp;248611 を参照してください。

シナリオ	JTAPI の動作
アプリケーションが JTAPI の getPrecedenceValue() API を使用して、CiscoRTPOutputStartedEvent で新しい DSCP 値を問い合せる。	JTAPI は StartTransmissionEvent で CTI から受信した DSCP 値をアプリケーションに返す。
アプリケーションが ProviderInServiceEvent を受信し、JTAPI の getAppDSCPValue() API を使用して新しい DSCP 値を問い合せる。	JTAPI は ProviderOpenCompletedEvent で CTI から受信した DSCP 値をアプリケーションに返す。

TLS セキュリティ

証明書を更新して TLS 認証を確立するためのメッセージ フローを、図 A-2 および 図 A-3 に示します。

図 A-2　CTI/API のセキュリティ アプローチ

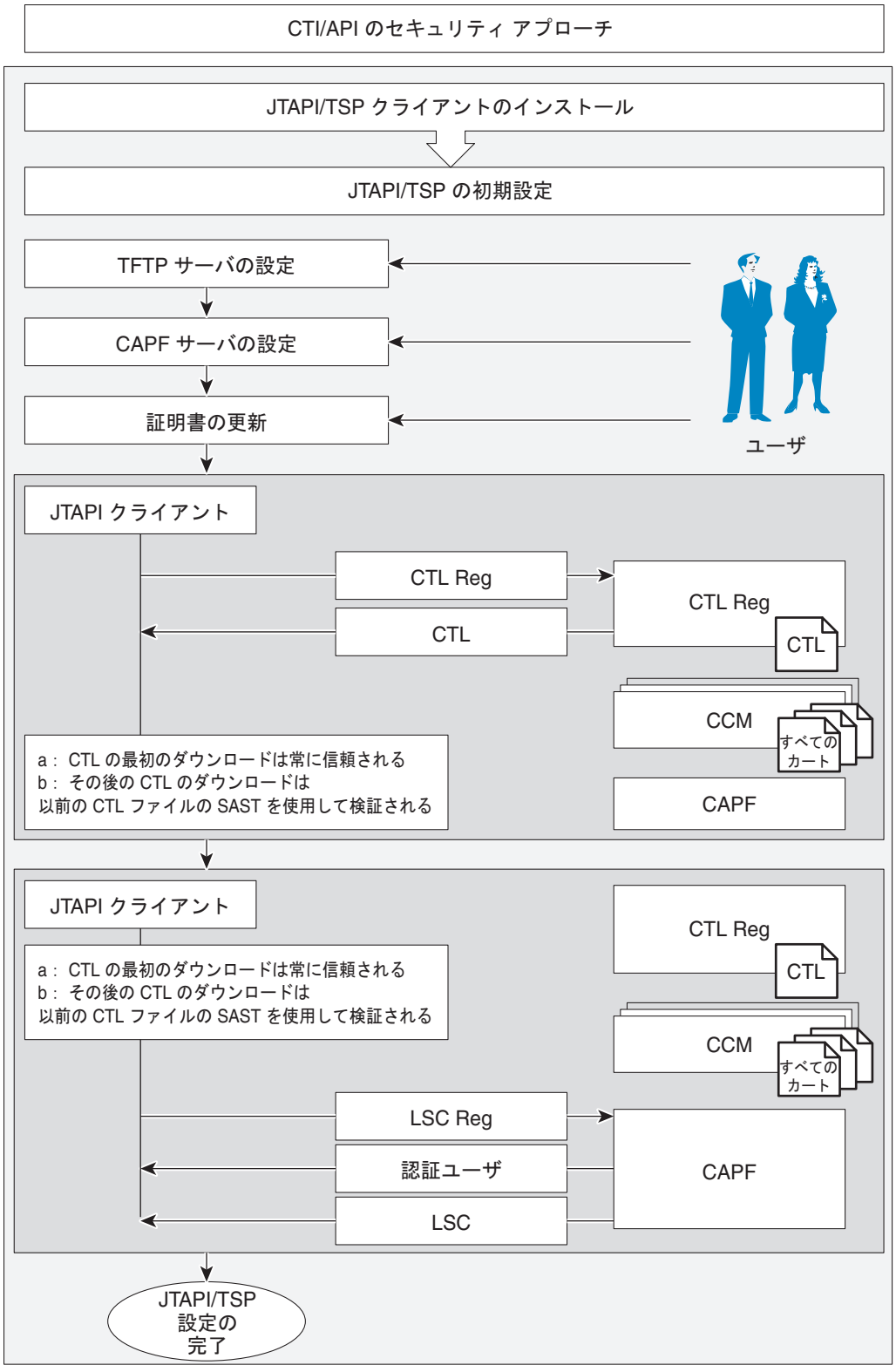
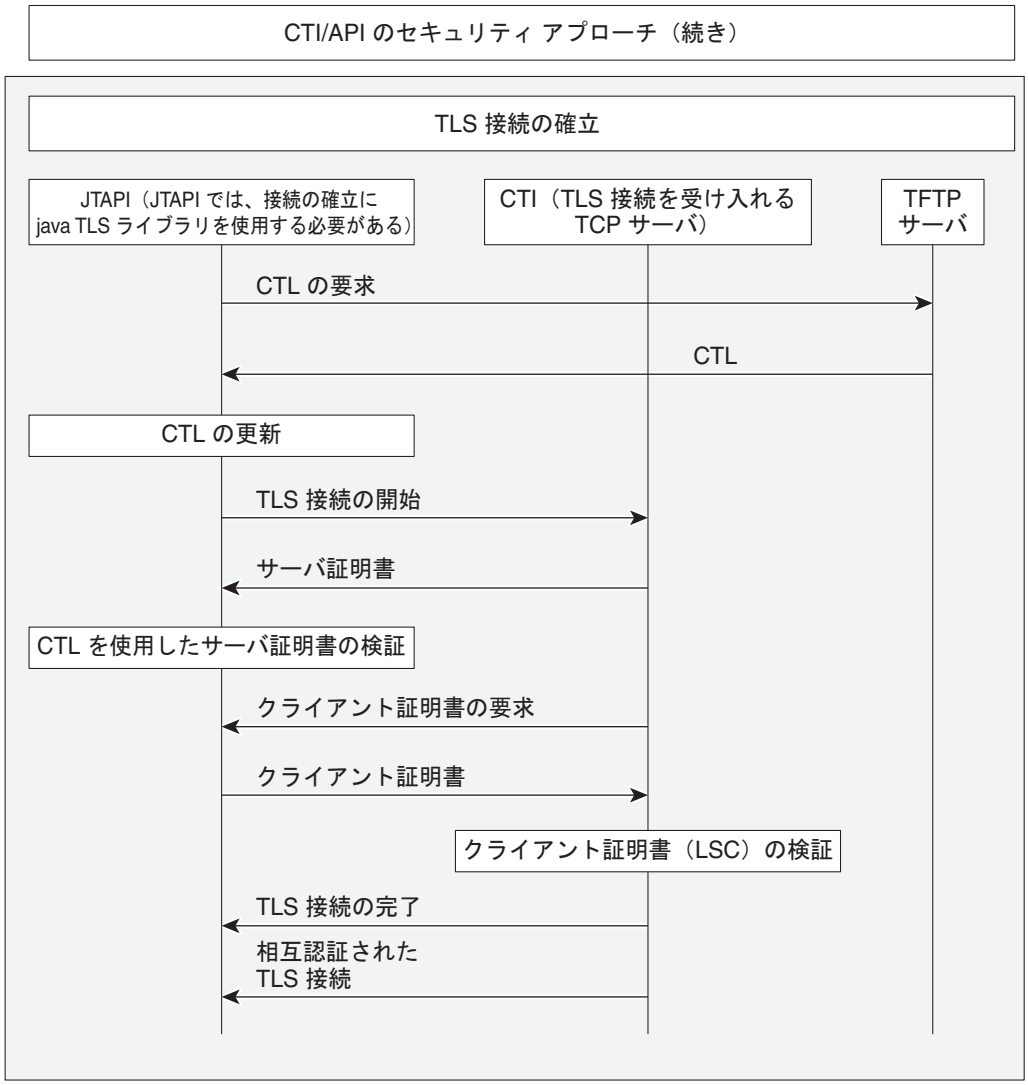


図 A-3　CTI/API のセキュリティ アプローチ（続き）



SRTP 鍵情報

この機能を有効にした場合、Cisco Unified JTAPI のパフォーマンスが低下することが予想されます。このパフォーマンス低下は、CTI と JTAPI の間での暗号化シグナリング、およびエンド ポイント間での暗号化メディアによるものです。

シナリオ 1

操作	イベント
アプリケーションが CallObserver をアドレス 1 に追加し、アドレス 2 へのコールを開始して、セキュアなメディア対話を実行する。 ユーザが権限を持っている場合は、CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv に鍵情報が含まれる。	CiscoRTPInputKeyEv CiscoRTPInputStartedEv CiscoRTPOutputKeyEv CiscoRTPOutputStartedEv

シナリオ 2

操作	イベント
アプリケーションが snapshotEnabled フィルタを有効にして TerminalObserver を追加する。デバイスはすでにセキュアなコールを行っており、問い合せを行うと CiscoTerminal.createSnapshot () が呼び出される。	アプリケーションは CiscoTermSnapshotEv を使用し、getCiscoMediaCallSecurity () に問い合せて、コールのセキュリティが確保されているかどうかを確認できる。

シナリオ 3

操作	応答
アプリケーションは TLS リンクを持っておらず、セキュアなメディアへの登録を試みる。CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm)	アプリケーションに PrivilegeViolationException がスローされる。
アプリケーションがセキュアなメディアを持っており、CiscoMediaTerminal.register (ipAddr, portNum, mediaCaps, algorithm) を登録する。	要求が成功する。

デバイスと回線の制限

番号	シナリオ	イベント
1	アプリケーションがデバイス T1、T2、T3 を持っており、その回線 A1、A2、A3 がコントロール リストに含まれている。T1 および A3 が制限リストに追加される。アプリケーションがプロバイダーをオープンする。	
	アプリケーションが T1、T2、T3 の isRestricted を問い合せる。	CiscoTerminal.isRestricted() が、T1 には true、T2 および T3 には false を返す。
	アプリケーションがアドレス A1、A2、A3 の isRestricted を問い合せる。	CiscoAddress.isRestricted() が、A1 および A3 には true、A2 には false を返す。
		CiscoAddress.getRestrictedAddrTerminals() は、A1 および A3 にはそれぞれ T1 および T3 を、A2 には null を返す。
	アプリケーションが T1、T2、T3、および A1、A2、A3 に addObserver および addCallObserver の実行を試みる。	T1、A1、A3 では、addObserver および addCallObserver が失敗する。T3 にはオブザーバが追加されるが、A3 ではイベントが受信されない。A2 にはアプリケーションが正常にオブザーバを追加でき、イベントが受信される。
2	アプリケーションがデバイス T1、T2、T3 を持っており、その回線 A1、A2、A3 がコントロール リストに含まれている。	
	アプリケーションがプロバイダーをオープンして、すべての端末およびアドレスにオブザーバを追加する。	T1 には CiscoTermRestrictedEv、L1 には CiscoAddrRestrictedEv A2 には CiscoAddrRestrictedEv が providerObserver に送信される。
	T1 および A2 が制限リストに追加される。	T1 には CiscoTermOutOfServiceEv、L1 には CiscoAddrOutOfServiceEv A2 には CiscoAddrOutOfServiceEv
	T1 および L2 が制限リストから削除される。	T1 には CiscoTermActivatedEv、A1 には CiscoAddrActivatedEv A2 には CiscoAddrActivatedEv が providerObserver に送信される。 T1 には CiscoTermInServiceEv、A1 には CiscoAddrInServiceEv A2 には CiscoAddrInServiceEv が端末およびアドレスのオブザーバに送信される。

3	アプリケーションがデバイス T1、T2、T3 を持っており、その回線 A1、A1、A2 がコントロール リストに含まれている。A1 は T1 および T2 上の共用回線。 アプリケーションがプロバイダーをオープンして、すべての端末およびアドレスにオブザーバを追加する。	
	T1 が制限リストに追加される。	アプリケーションは、T1 について CiscoTermRestrictedEv を受信し、さらに L1 として getAddress、T1 として getTerminal を含む CiscoAddrRestrictedOnTerminalEv を受信する。また、T1 について CiscoTermOutOfServiceEv、A1 および T1 について CiscoAddrOutOfService も受信する。
	T1 が制限リストから削除される。	T1 には CiscoTermActivatedEv L1 には CiscoAddrActivatedEv T1 には CiscoTermInServiceEv A1/T1 には CiscoAddrInServiceEv
4	アプリケーションがデバイス T1、T2、T3 を持っており、その回線 A1、A1、A1 がコントロール リストに含まれている。A1 は T1、T2、および T3 上の共用回線。 アプリケーションがプロバイダーをオープンして、すべての端末およびアドレスにオブザーバを追加する。	
	T1 の A1 が制限リストに追加される。	A1/T1 には CiscoAddrRestrictedOnTerminalEv A1/T1 には CiscoAddrOutOfServiceEv
	T2 の A1 が制限リストに追加される。	A1/T1 には CiscoAddrRestrictedOnTerminalEv A1/T2 には CiscoAddrOutOfServiceEv
	T3 の A1 が制限リストに追加される。	A1 には CiscoAddrRestrictedEv A1/T3 には CiscoAddrOutOfServiceEv
	T1 の A1 が制限リストから削除される。	A1/T1 には CiscoAddrActivatedOnTerminalEv A1/T1 には CiscoAddrInServiceEv
	T2 の A1 が制限リストから削除される。	A1/T2 には CiscoAddrActivatedOnTerminalEv A1/T2 には CiscoAddrInServiceEv
	T3 の A1 が制限リストから削除される。	A1 には CiscoAddrActivatedEv A1/T3 には CiscoAddrInServiceEv

5	アプリケーションがデバイス T1、T2、T3 を持っており、その回線 A1、A2、A3 がコントロール リストに含まれている。	
	アプリケーションがプロバイダーをオープンして、すべての端末およびアドレスにオブザーバを追加する。 A1 は通話者 X とのコールに参加する。 A1 が制限リストに追加される。	A1 には CiscoAddrRestrictedEv A1 には CiscoAddrOutOfServiceEv ConnDisconnectedEv CallCtlConnDisconnectedEv TermConnDroppedEv CallCtlConnDroppedEv CallInvalidEv

SIP のサポート

番号	シナリオ	イベント
1	外部の SIP フォン（external@someserver.com）が A にコールする。A はアプリケーションによって監視されている。 外部 SIP フォンは、DN ではなく URI を使用するものとする。	A のコール オブザーバに配信されるイベント CallActiveEv ConnCreatedEv A ConnCreatedEv unknown getCurrentCallingPartyInfo().geUrlInfo().getUser() が external を返す。 getCurrentCallingPartyInfo().geUrlInfo().getHost() が someserver.com を返す。 getCurrentCallingPartyInfo().geUrlInfo().getUrlType() が SIP_URL_TYPE を返す。
2	7970 が、最大コール数 2 に設定された SIP プロトコルを実行する。3 番目のコールは GCID=GCID3 で着信する。	GCID3 CallActiveEv GCID3 ConnCreatedEv A GCID3 ConnFailedEv A GCID3 callInvalidEv
3	SIP を実行する 7960 はコントロールリストに含まれている。アプリケーションが端末にコール オブサーバを追加する。	addobserver 例外に例外がスローされる。ステータスが変更されると、TerminalRestrictedEv が配信される。

SIP REPLACE

次に説明する各シナリオの JTAPI イベントでは、Terminal イベントは示していません。Terminal イベントは、監視されているすべての Terminal に通常どおり送信されます。またイベントは、A、B、または C のいずれかだけが監視されるという前提で示されています。A、B、C が組み合わせて監視されている場合はイベントが変わります。

番号	シナリオ	A のイベント	B のイベント	C のイベント
1.	confirmed ダイアログを INVITE で置換する： A（Dialog1）は B（Dialog2）とのコール中である（GC1）。C が REPLACE Dialog2 を含む INVITE を送信する（GC2）。置換が完了すると、A（Dialog1）および C（Dialog3）がコール中になる。	原因が REPLACES の GCID および CPIC、Cgpn=C, Cdpn=A, Ocdpn=A, Lrp=B JTAPI イベント： CiscoCallChangedEv-(GC1-GC2) ConnDisconnectedEv –B-GC1 CallCtlConnDisconnectedEv-B–GC1 ConnDisconnectedEv –A-GC1 CallCtlConnDisconnectedEv-A–GC1 CallInvalid-GC1 CallActive-CG2 ConnCreatedEv –C-GC2 ConnConnectedEv –C-GC2 CallCtlConnEstablishedEv-C-CG2_ ConnCreatedEv –A –GC2 ConnConnectedEv –A-GC2 CallCtlConnEstablishedEv-A-CG2 Cause =CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B	原因が REPLACES の CSCE IDLE JTAPI イベント： ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv-A-GC1 ConnDisconnectedEv –B-GC1 CallCtlConnDisconnectedEv-B-GC1 CallInvalidEv-GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES	NewCall/CSCE-Dialing/ CSCE-Connected with Cgpn=C, Cdpn=A, Ocdpn=B, Lrp=B JTAPI イベント： CallActiveEv –GC2 ConnCreatedEv –C –GC2 ConnConnectedEv-C-–GC2 CallCtlConnEstablishedEv-C–GC2 ConnCreatedEv –A–GC2 ConnConnectedEv A-–GC2 CallCtlConnEstablishedEv-A-–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B

2.	early ダイアログを INVITE で置換する： A（Dailog1）は B（Dialog2）とのコール中であり（GC1）、B が呼び出し中になる。C が REPLACE Dialog2 を含む INVITE を送信する（GC2）。置換が完了すると、A（Dialog1）および C（Dialog3）がコール中になる。	原因が REPLACES の GCID および CPIC、Cgpn=C, Cdpn=A, Ocdpn=A, Lrp=B JTAPI イベント CiscoCallChangedEv-(GC1-GC2) ConnDisconnectedEv –B-GC1 CallCtlConnDisconnected Ev-B–GC1 ConnDisconnectedEv –A-GC1 CallCtlConnDisconnected Ev-A–GC1 CallInvalid-GC1 CallActive-CG2 ConnCreatedEv –C-GC2 ConnConnectedEv –C-GC2 CallCtlConnEstablishedEv-C-CG2_ ConnCreatedEv –A –GC2 ConnConnectedEv –A-GC2 CallCtlConnEstablishedEv-A-CG2 Cause =CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B	原因が REPLACES の CSCE-Idle JTAPI イベント： ConnDisconnectedEv –A –GC1 CallCtlConnDisconnected Ev-A-GC1 ConnDisconnectedEv –B-GC1 CallCtlConnDisconnected Ev-B-GC1 CallInvalidEv-GC1 CAUSE_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES	NewCall/CSCE-Dialing/CSCE-Connected、Cgpn=C, Ccdpn=A, Ocdpn=A, Lrp=B JTAPI イベント： CallActiveEv –GC2 ConnCreatedEv –C –GC2 ConnConnectedEv-C-–GC2 CallCtlConnEstablishedEv-C–GC2 ConnCreatedEv –A–GC2 ConnConnectedEv A-–GC2 CallCtlConnEstablishedEv-A-–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=A, CurrentCalling=C, CurrentCalled=A, LastRedirecting=B

3.	early ダイアログを INVITE で置換する： A（Dialog1）は B（Dialog2）とのコール中であり（GC1）、B が呼び出し中になる。C が REPLACE Dialog-X を含む INVITE を送信する（GC2）。			原因が REPLACES の NewCall/CSCE_Dialing/ 原因が REPLACES の CSCE-Disconnected JTAPI イベント： CallActiveEv –GC2 ConnCreatedEv –C–GC2 ConnConnectedEv –C–GC2 CallCtlConnEstablishedEv-C-–GC2 ConnFailedEv –C-–GC2 ConnConnectedEv –C-–GC2 CallCtlConnEstablishedEv-A-–GC2 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=, CurrentCalling=C, CurrentCalled=, LastRedirecting=
4.	REPLACE ダイアログによる REFER 要求： REPLACE ダイアログが Cisco Unified Communications Manager クラスタ内にある場合： A は B（Referee）とのコール中 Dialog1 および Dialog2 A は C（Refer To Target）とのコール中 Dialog3 および Dialog4 SIP-UA A が Dialog1 で、REPLACES Dialog3 を含む REFER B を C に送信する。	TransferStartEv Dialog1 で原因が TRANSFER の CSCE-Idle、および Dialog3 で原因が TRANSFER の CSCE-Idle TransferEndEv JTAPI イベント： 通常の TransferEvent	TransterStartEv 原因が TRANSFER の CPIC、Cgpn=B, Cdpn=C, Lrp=A OCdpn=C TransferEndEv JTAPI イベント： 通常の TransferEvent	TransferStartEv 原因が TRANSFER の GCID、Cgpn=B, Cdpn=C, Lrp=A OCdpn=C TransferEndEv JTAPI イベント： 通常の TransferEvent

5.	REPLACE ダイアログによる REFER 要求： REPLACE ダイアログが Cisco Unified Communications Manager クラスタの外にある場合： SIP-UA A は B とのコール中、Dialog1 および Dialog2（GC1） SIP-UA A は SIP-UA C とのコール中、Dialog3 SIP-UA A が Dialog1 で、REPLACES Dialog3 を含む REFER B を SIP-UA C に送信する。	イベントなし	原因が REFER の CPIC、Cgpn=B, Cdpn=C, Lrp=A OCdpn=B JTAPI イベント： ConnDisconnectedEv –A-GC1 CallCtlConnDisconnectedEv-A-GC1 ConnCreatedEv – C –GC1 ConnConnectedEv –C-GC1 CallCtlConnEstablishedEv –C-GC1 Cause = CAUSE_NORMAL CiscoFeatureReason=REASON_REFER JTAPI CallInfo： Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A	イベントなし

6.	REPLACE ダイアログによる REFER 要求： A が Cisco Unified Communications Manager クラスタの外にある場合： SIP-UA A は B とのコール中、Dialog1 および Dialog2 SIP-UA A は C とのコール中、Dialog3 および Dialog4 SIP-UA A が Dialog1 で、REPLACES Dialog3 を含む REFER B を C に送信する。	イベントなし	原因が REPLACES の CPIC、Cgpn=B, Cdpn=C, Lrp=A, OCdpn=C JTAPI イベント： ConnDisconnectedEv –A-GC1 CallCtlConnDisconnectedEv-A-GC1 ConnCreatedEv – C- GC1 ConnConnectedEv –C –GC1 CallCtlConnEstablishedEv –C-GC1 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A	原因が REPLACES の GCID、Cgpn=B, Cdpn=C, Lrp=A OCdpn=C JTAPI イベント： CiscoCallChangedEv(GC2-GC1) ConnDisconnectedEv –A-GC2 CallCtlConnDisconnectedEv-A-GC2 ConnDisconnectedEv –C-GC2 CallCtlConnDisconnectedEv-C-GC2 CallInvalid-GC2 CallActive-CG1 ConnCreatedEv –B –GC1 ConnConnectedEv –B-GC1 CallCtlConnEstablishedEv-B-CG1 ConnCreatedEv –C-GC1 ConnConnectedEv –C-GC1 CallCtlConnEstablishedEv-C-CG1 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=A

7.	REPLACE ダイアログによる REFER 要求： REPLACE ダイアログが Cisco Unified Communications Manager クラスタ内にある場合： A は B（Referee）とのコール中 Dialog1 および Dialog2（GC1） D は C（Refer To Target）とのコール中 Dialog3 および Dialog4（GC2） A が Dialog1 で、REPLACES Dialog3 を含む REFER B を C に送信する。 B と C は最後のコール中。	Dialog1 で原因が REFER の CSCE-Idle、および Dialog3 で原因が REPLACES の CSCE-Idle JTAPI イベント： ConnDisconnectedEv –A –GC1 CallCtlConnDisconnectedEv-A-GC1 ConnDisconnectedEv –B-GC1 CallCtlConnDisconnectedEv-B-GC1 CallInvalidEv-GC1 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REFER D のイベント： ConnDisconnectedEv –D –GC2 CallCtlConnDisconnectedEv-D-GC2 ConnDisconnectedEv –C-GC2 CallCtlConnDisconnectedEv-C-GC2 CallInvalidEv-GC2 Cause = CAUSE_NORMAL CiscoFeatureReason=REASON_REPLACES	原因が REPLACES の CPIC、Cgpn=B, Cdpn=C, Lrp=D OCdpn=C JTAPI イベント： ConnDisconnectedEv –A-GC1 CallCtlConnDisconnectedEv-A-GC1 ConnCreatedEv –C-GC1 ConnConnectedEv –C –GC1 CallCtlConnEstablishedEv-C-GC1 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=A, Called=B, CurrentCalling=B, CurrentCalled=C, LastRedirecting=D	原因が REPLACES の GCID、Cgpn=B, Cdpn=C, Lrp=D, OCdpn=C JTAPI イベント： CiscoCallChangedEv(GC2-GC1) ConnDisconnectedEv -D CallCtlConnDisconnectedEv-D ConnDisconnectedEv -C CallCtlConnDisconnectedEv-C CallInvalid-GC2 CallActive-CG1 ConnCreatedEv –C-GC1 ConnConnectedEv –C-GC1 CallCtlConnEstablishedEv-C-CG1 ConnCreatedEv –B –GC1 ConnConnectedEv –B-GC1 CallCtlConnEstablishedEv-B-CG2 Cause = CAUSE_NORMAL CiscoFeatureReason= REASON_REPLACES JTAPI CallInfo： Calling=C, Called=C, CurrentCalling=B, CurrentCalled=C, LastRedirecting=D

SIP REFER

次のセクションでは、SIP REFER の実行時に発生する可能性があるシナリオを説明します。REFER シナリオには、IN-Dialog および OutOfDialog の 2 つのカテゴリがあります。

IN-Dialog REFER シナリオ

次の IN-Dialog REFER に関する各セクションでは、11 のシナリオ（A ～ K）を説明します。

シナリオ 1

A（クラスタ内または制御内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になる。

JTAPI が A の Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection を「UNKNOWN」状態にする。

提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

C には、新しい Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection が作成される。

B および C の CallInfo は次のようになる。

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

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

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

シナリオ 2

A（クラスタ内または制御内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C がコールに応答する。

JTAPI は A の Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection を接続解除またはドロップする。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

C では、Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection が、Connected/Established/Active/Talking 状態に移行する。

B および C の CallInfo は次のようになる。

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

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

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

シナリオ 3

A（クラスタ内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になるが、C はコールに応答せず、転送設定も行われていない。参照は失敗し、A と B 間の元のコールが再開される。

JTAPI は C の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を接続解除またはドロップする。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供し、A の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を「Unknown」状態から Connected/Established/Active/Talking 状態に移行させる。

A および B の CallInfo は次のようになる。

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

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

A を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()= NULL

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()=NULL

シナリオ 4

A（クラスタ外の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になる。

JTAPI は C の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を作成し、B で CPIC を取得する際に A の Connection/CallControlConnection をドロップする。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

B および C の CallInfo は次のようになる。

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

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

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

シナリオ 5

A（クラスタ外の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になるが、C はコールに応答せず、転送設定も行われていない。参照は失敗し、A と B 間の元のコールが再開される。

JTAPI は再度 A の Connection/CallControlConnection を作成し、C の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection をドロップする。
提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

A および B の CallInfo は次のようになる。

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

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

A を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()=NULL

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=A

getCurrentCalledParty()=B

getLastRedirecting()=NULL

シナリオ 6

A（クラスタ内または制御内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C がコールに応答する。

JTAPI は C の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を、Connected/Established/Active/Talking 状態に移行させる。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

B および C の CallInfo は次のようになる。

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

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

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

シナリオ 7

A（クラスタ内または制御内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C は D への forwardAll を実行して、D が呼び出し中になる。

JTAPI は D の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を作成する。提供される CAUSE_CODE は CAUSE_REDIRECT で、CTI から受信される原因は ForwardAll になる。

B および D の CallInfo は次のようになる。

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

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

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=D

getLastRedirecting()=C

D を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty()=B

getCurrentCalledParty()=D

getLastRedirecting()=C

シナリオ 8

A（クラスタ内または制御内の SIP UA）は B とのコール中。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C は D へのリダイレクトを実行し、D が呼び出し中になる。

JTAPI は D の Connection/CallControlConnection/TerminalConnection/CallControlTerminalConnection を作成する。提供される CAUSE_CODE は CAUSE_REDIRECT で、D の NewCallEvent で CTI から受信される原因は Redirect になる。

C にコールが提供されたときの Callinfo：

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

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

最後のコールの CallInfo：

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

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

B を監視している JTAPI アプリケーションに最後のコールで示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=D

getLastRedirecting()=C

D を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty()=B

getCurrentCalledParty()=D

getLastRedirecting()=C

シナリオ 9

A（クラスタ内または制御内の SIP UA）は B とのコール中。
B が D へのコンサルト転送を行い、A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になって、B による転送が成立する。C の呼び出し中は、転送が失敗する。

シナリオ 10

A（クラスタ内または制御内の SIP UA）は B とのコール中。
B が D へのコンサルト転送を行い、A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C がコールに応答する。
参照が成功する。B が転送を実行し、転送は成功して C と D がコール中になる。

JTAPI は A の Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection を接続解除またはドロップする。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。

C では、Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection が、Connected/Established/Active/Talking 状態に移行する。

D および C の CallInfo は次のようになる。

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

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

D を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty()=C

getCurrentCalledParty()=D

getLastRedirecting()=B

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=C

getCurrentCalledParty()=D

getLastRedirecting()=B

シナリオ 11

B は D とのコール中。B は A（クラスタ内または制御内の SIP UA）にコンサルト コールする。
A（Referrer）が B（Referee）に C（Refer To Target）を参照させ、C が呼び出し中になって、B による転送が成立する。
REFER は失敗する。A のコールはドロップされ、転送が成功し、D は RingBack を受け取って、C が呼び出し中になる。

JTAPI は A の Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection を接続解除またはドロップする。提供される CAUSE_CODE は CAUSE_NORMAL で、新しい API が REASON_REFER を提供する。アプリケーションには、REFER が失敗したかどうかは知らされない。

C では、Connect/CallControlConnection/TerminalConnection/CallControlTerminalConnection が、Alerting/Alerting/Ringing/Ringing 状態に移行する。

D および C の CallInfo は次のようになる。

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

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

D を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = D

getCurrentCallingParty()=D

getCurrentCalledParty()=C

getLastRedirecting()=B

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

getCurrentCallingParty()=D

getCurrentCalledParty()=C

getLastRedirecting()=B

OutOfDialog Refer

SIP-UA A が B（Referee）に C（Refer To Target）を参照させる。
B は Cgpn=A、Cdpn=B、Lrp=、OCdpn=B の newcall を受け取る。

JTAPI アプリケーションは B に作成された CallActive、Connection、CallCtlConnection、TerminalConnecton、および CallCtlTerminalConnection を CAUSE_NORMAL とともに受信し、新しい API が REASON_REFER を返す。

B の Connection/CallCtlConnection および TerminalConnection/CallCtlTerminalConnection は、Connected/Established/Active/Talking 状態に移行する。JTAPI は、CTI/CP に提供された FarEndPointType_ServerCall に基づいて、A に「UNKNOWN」状態の Connection および CallCtlConnection を作成する。

B がコールに応答し、A との接続が確立される（この段階では RTP イベントは送信されない）。

B が、Cgpn=B、Cdpn=C、Lrp=A、OCdpn=C、Reason=REFER の CallPartyInfoChangedEv を受信する。

C が、Cgpn=B、Cdpn=C、Lrp=A、OCdpn=C、Reason=REFER を提供する NewCall を受信する。

JTAPI アプリケーションは B に作成された Connection、CallControlConnection、TerminalConnecton、および CallCtlTerminalConnection を CAUSE_NORMAL とともに受信し、新しい API が REASON_REFER を返す。

C がコールを受け入れるか応答し、B が C と接続される（これでアプリケーションが RTP イベントを受信する）。

C の Connection/CallCtlConnection および TerminalConnection/CallCtlTerminalConnection は、Connected/Established/Active/Talking 状態に移行する。

B を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = A

getCalledParty() = B

getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

C を監視している JTAPI アプリケーションに示される内容：

getCallingParty() = B

getCalledParty() = C

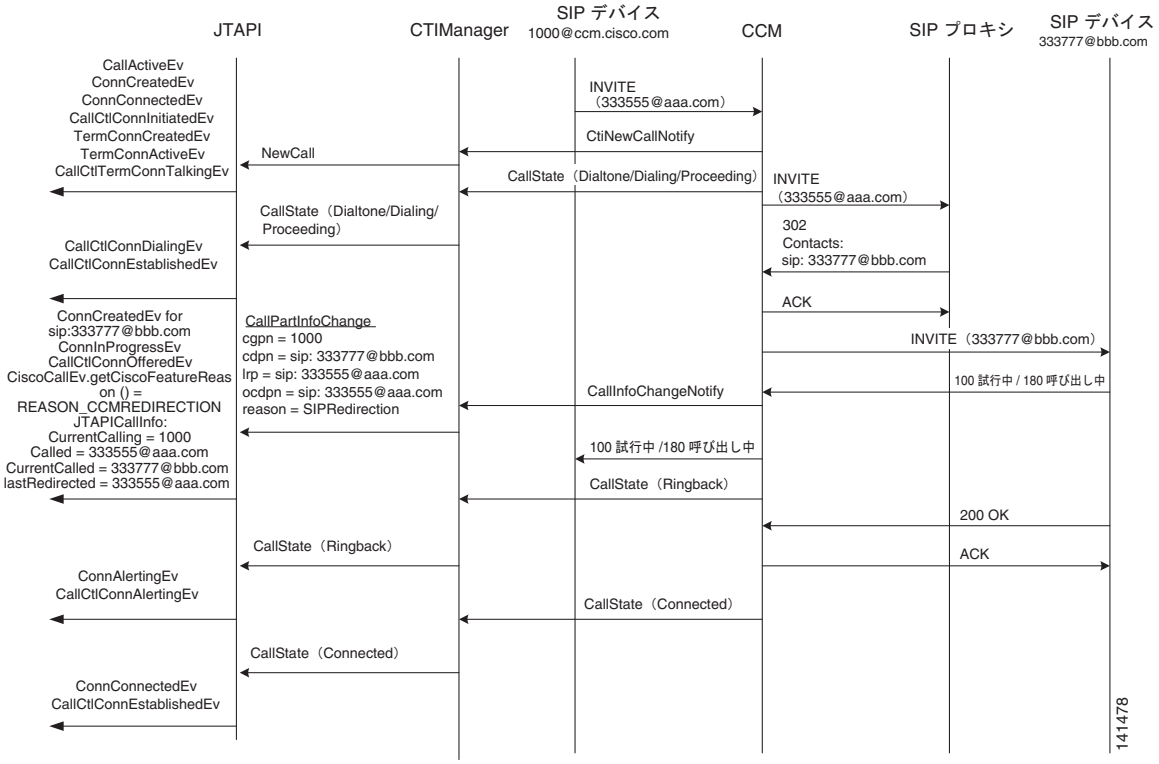
getCurrentCallingParty()=B

getCurrentCalledParty()=C

getLastRedirecting()=A

SIP 3XX リダイレクション

3XX リダイレクション：302 Moved Temporarily



JTAPI アプリケーションが 1000@ccm.cisco.com を監視している。

Cisco Unified Communications Manager の user1000 が、333555@aaa.com へのコールを開始する。

CTI が INVITE に基づいて NewCallNotify および CtiCallStateNotify（Dialtone/Dialing）を報告する。

JTAPI が、CallActiveEv、および 1000 の Connection イベントと CallCtlConnection イベントを報告する。

JTAPI が CallCtlConnEstablishedEv を報告する。

SIP プロキシが 333555@aaa.com の 302 を報告する。302 に基づいて、Cisco Unified Communications Manager が 333777@bbb.com への q 値に基づくターゲット リストに含まれる最初の接点へのコールを開始する。

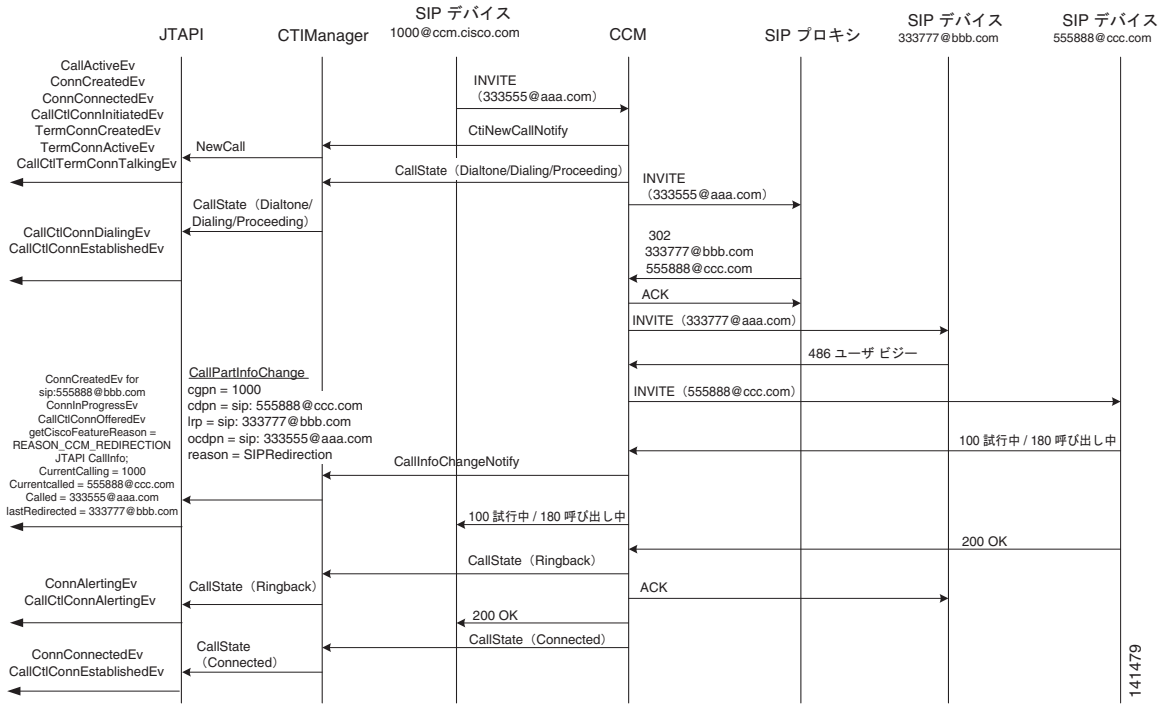
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 333777@bbb.com への接続作成イベントを報告する。

CTI が CtiCallStateNotify（Ringback）および CtiCallStateNotify（Connected）を報告する。

JTAPI が遠端の ConnAlertingEv および ConnEstablishedEv を報告する。

3XX リダイレクション：Contact Busy



JTAPI CTI アプリケーションが 1000@ccm.cisco.com を監視している。

Cisco Unified Communications Manager の user1000 が、333555@aaa.com へのコールを開始する。

CTI が INVITE に基づいて NewCallNotify および CtiCallStateNotify（Dialtone/Dialing）を報告する。

JTAPI が、CallActiveEv、および 1000 の Connection イベントと CallCtlConnection イベントを報告する。

CTI が CtiCallStateNotify（Proceeding）を報告する。

JTAPI が CallCtlConnEstablishedEv を報告する。

SIP プロキシが 333555@aaa.com の 302 を報告する。302 に基づいて、Cisco Unified Communications Manager が 333777@bbb.com への q 値に基づくターゲット リストに含まれる最初の接点へのコールを開始する。

333777@bbb.com から、486 ユーザ ビジー応答が報告される。この応答に基づいて、Cisco Unified Communications Manager が 555888@cisco.com へのコールを開始する。

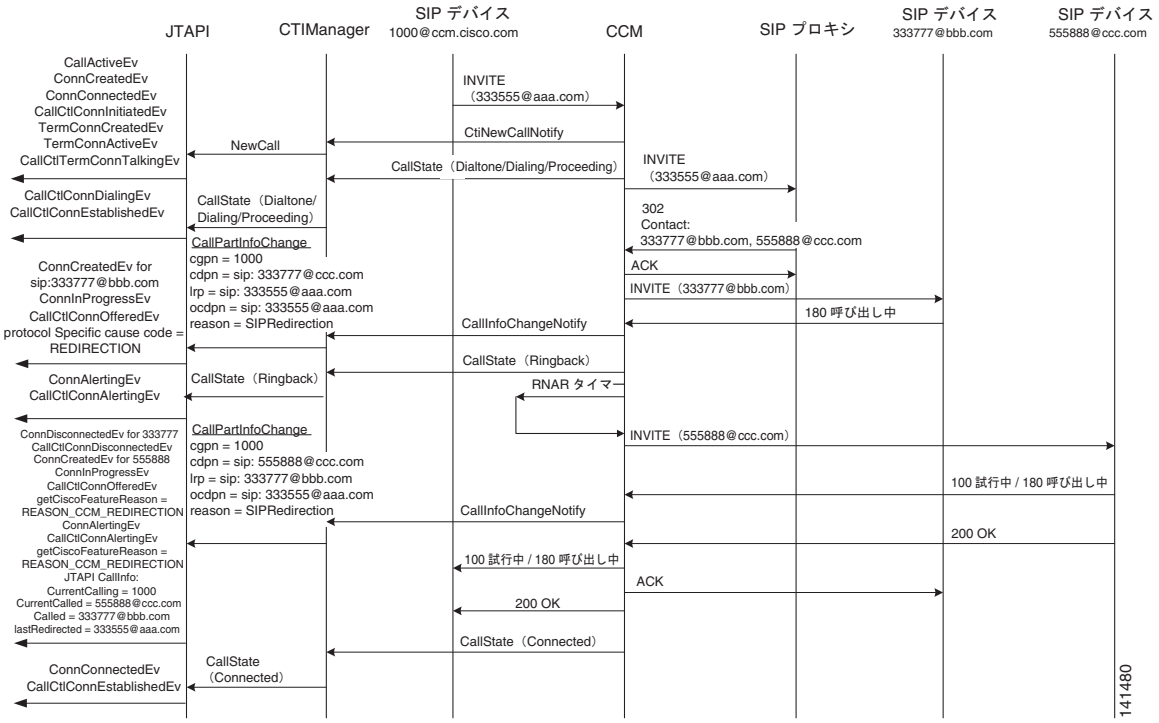
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 555888@cisco.com への接続作成イベントを報告する。

CTI が、CtiCallStateNotify（Ringback）および CtiCallStateNotify（Connected）も報告する。

JTAPI が新しい通話者の CallCtlConnAlertingEv および CallCtlConnEstablishedEv を報告する。

3XX リダイレクション：Contact Does Not Answer



JTAPI アプリケーションが 1000@ccm.cisco.com を監視している。

Cisco Unified Communications Manager の user1000 が、333555@aaa.com へのコールを開始する。

CTI が INVITE に基づいて NewCallNotify および CtiCallStateNotify（Dialtone/Dialing）を報告する。

JTAPI が、CallActiveEv、および 1000 の connection イベントと terminalConnection イベントを報告する。

CTI が CtiCallStateNotify（Proceeding）を報告する。

JTAPI が 1000 の CallCtlConnEstablishedEv を報告する。

SIP プロキシが 333555@aaa.com の 302 を報告する。302 に基づいて、Cisco Unified Communications Manager が 333777@bbb.com への q 値に基づくターゲット リストに含まれる最初の接点へのコールを開始する。Cisco Unified Communications Manager が RNAR タイマーを開始する。

着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 333777 への接続作成イベントを報告する。

RNAR タイマーが満了し、これに基づいて Cisco Unified Communications Manager が 555888@cisco.com へのコールを開始する。

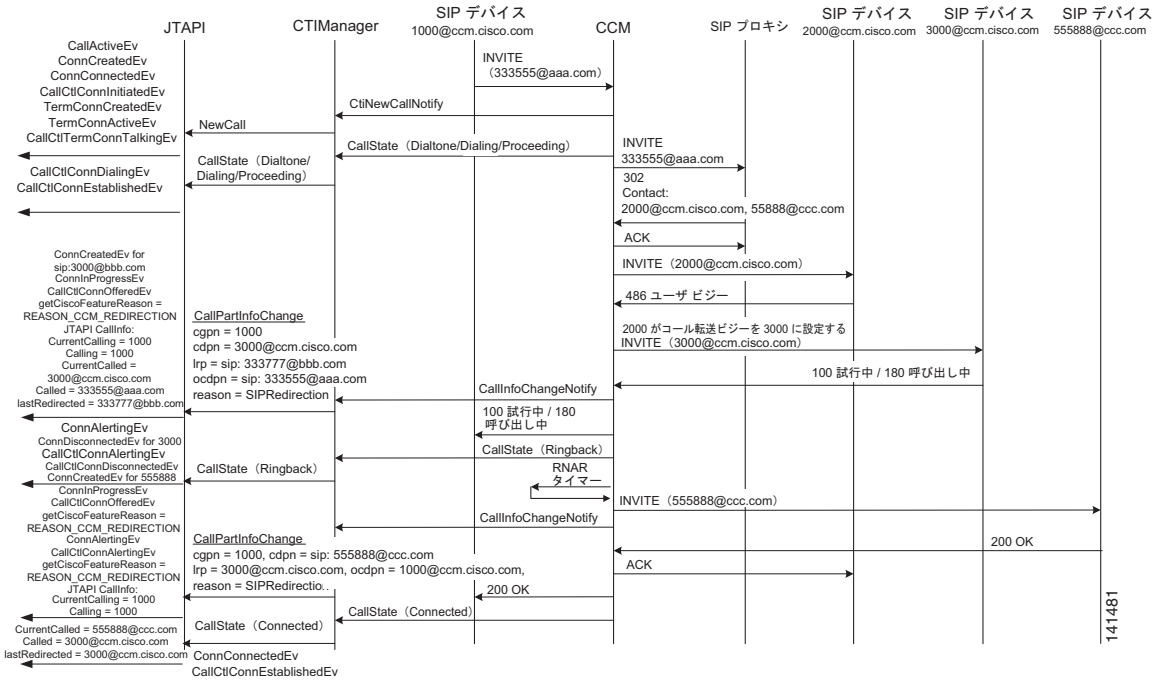
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd/CcNotifyReq に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 333777 への接続を削除し、555888 への接続を作成する。

CTI は CtiCallStateNotify（Connected）も報告する。

JTAPI が 555888 の CallCtlConnEstablishedEv を報告する。

3XX リダイレクション : Contact Within Cisco Unified Communications Manager Cluster Configured with Call Forward



JTAPI アプリケーションが 1000@ccm.cisco.com を監視している。

Cisco Unified Communications Manager の user1000 が、333555@aaa.com へのコールを開始する。

CTI が INVITE に基づいて NewCallNotify および CtiCallStateNotify（Dialtone/Dialing）を報告する。

JTAPI が、CallActiveEv、および 1000 の connection イベントと terminalConnection イベントを報告する。

CTI が CtiCallStateNotify（Proceeding）を報告する。

JTAPI が 1000 の CallCtlConnEstablishedEv を報告する。

SIP プロキシが 333555@aaa.com の 302 を報告する。302 に基づいて、Cisco Unified Communications Manager が 2000@ccm.cisco.com への q 値に基づくターゲット リストに含まれる最初の接点へのコールを開始する。

2000@ccm.cisco.com から、486 ユーザ ビジー応答が報告される。2000 にはコール転送ビジーが設定されているため、Cisco Unified Communications Manager が 3000@ccm.cisco.com へのコールを開始する。Cisco Unified Communications Manager は RNAR タイマーも開始する。

着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 3000 への接続作成イベントを報告する。

3000 は応答せず、RNAR タイマーが満了し、これに基づいて Cisco Unified Communications Manager が 555888@cisco.com へのコールを開始する。

着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd/CcNotifyReq に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 3000 への接続を破棄し、555888 への接続を作成する。

CTI は CtiCallStateNotify（Connected）も報告する。

JTAPI が 555888 の CallCtlConnEstablishedEv を報告する。

3XX リダイレクション - Non-Available Target Member

JTAPI アプリケーションが 1000@ccm.cisco.com を監視している。

Cisco Unified Communications Manager の user1000 が、333555@aaa.com へのコールを開始する。

CTI が INVITE に基づいて NewCallNotify および CtiCallStateNotify（Dialtone/Dialing）を報告する。

JTAPI が、CallActiveEv、および 1000 の connection イベントと terminalConnection イベントを報告する。

CTI が CtiCallStateNotify（Proceeding）を報告する。

JTAPI が 1000 の CallCtlConnEstablishedEv を報告する。

SIP プロキシが 333555@aaa.com の 302 を報告する。302 には、1212@ccm.cisco.com および 2000@ccm.cisco.com のターゲット リストが含まれる。1212@ccm.cisco.com は無効な DN。Cisco Unified Communications Manager はまず 1212@ccm.cisco.com へのコンタクトを試みるが、無効な DN が返されるため、2000@ccm.cisco.com へのコールを実行する。

着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

JTAPI が 2000 への接続作成イベントを報告する。

CTI は CtiCallStateNotify（Ringback/Connected）も報告する。

JTAPI が 2000 の CallCtlConnAlertingEv および CallCtlConnEstablishedEv を報告する。

Unicode のサポート

Unicode 表示名シナリオ

シナリオ	JTAPI アプリケーションに配信されるイベント
IP フォン A に、ASCII 名がなく日本語の Unicode 名がある回線が設定されている。IP フォン B には、ASCII 名があり、Unicode 名は設定されていない。 A が B にコールする。B だけが監視されている。	コール情報には次のものが含まれます。 getCurrentCalledPartyDisplayName=asciiNameB getCurrentCalledPartyUnicodeDisplayName=null getCurrentCallingPartyDisplayName=null getCurrentCallingPartyUnicodeDisplayName= japaneseNameA
A、B、および C が会議中。	DisplayName は該当しません。アプリケーションは「conference」を着信者と見なす必要があります。
共用回線：A および B は共用回線で、ロケールが異なる。A が C にコールする。C は監視されていない。	発信者の Unicode 表示名は、A と B のいずれかに変更できます。

端末の GetLocale と UniCodeCapabilities

シナリオ	JTAPI アプリケーションに配信されるイベント
IP フォン A に、ASCII 名がなく日本語の Unicode 名がある回線が設定されている。	
アプリケーションがデバイスに TerminalObserver を追加する。	CiscoTerminalInServiceEv の内容： getLocale = JAPANESE getSupportedEncoding= UCS2UNICODE_ENCODING
アプリケーションが CiscoTerminal を使用して次の問い合せを行う。	CiscoTerminal.getLocale = JAPANESE CiscoTerminal.getSupportedEncoding= UCS2UNICODE_ENCODING

下位互換性に関する機能拡張

この機能は、Cisco Unified Communications Manager JTAPI のパフォーマンスやスケーラビリティを変えるためのものではありません。JTAPI と CTI のイベント数に違いはありません。GCID の変更を含む機能には、この機能によって追加イベントが 1 つ導入されます。このイベントは通常、パフォーマンス問題の原因にはなりません。

次に示す各イベントは、どのような場合でも、コントロール リストに 1 つの通話者だけが存在するときコール オブザーバに配信されます。TERMA は A の端末を示します。

シナリオ 1

A が B にコールし、B がそのコールを C に転送します。GC1 は A と B の間のコールであり、GC2 は B と C の間のコンサルト コールです。Conference やその他の機能についても同様のイベントが配信されます。

操作	イベント
B が転送を実行する。C のコール オブザーバに配信されるイベント	GC2 CiscoTransferStartEv Cause: CAUSE_NORMAL Reason=REASON_TRANSFER CallActiveEv GC1 Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER ConnCreatedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER ConnCreatedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER CiscoCallChangedEv SurvingCall=GC1, original call=GC2 CiscoCause: NORMAL Reason: REASON_TRANSFER

操作	イベント
B の CallObserver に配信されるイベント（転送コントローラ）	ConnConnectedEv C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER TermConnCreatedEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER TermConnActiveEv TERM C Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER CallCtlTermConnTalkingEv TERM C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER ConnConnectedEv B Cause: CAUSE_NORMAL CiscoCause: CAUSE_NORMALUNSPECIFIED Reason=REASON_TRANSFER GC2: ConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_NORMAL GC2: ConnDisconnectedEv C REASON=REASON_TRANSFER Cause: CAUSE_NORMAL GC2: TermConnDropped TERMB REASON=REASON_TRANSFER Cause: CAUSE_NORMAL GC2: CalInvalid REASON=REASON_TRANSFER Cause: CAUSE_NORMAL GC1: CallCtlTermConnHeldEv TERMB REASON=REASON_TRANSFER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC2: ConsultCallActive REASON=NORMAL Cause: CAUSE_NEW_CALL GC2: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC2: ConnConnectedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN

操作	イベント
B の CallObserver に配信されるイベント（転送コントローラ）（続き）	GC1: CallCtlConnDisconnectedEv B REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC1: TermConnDroppedEv TERMB REASON=REASON_TRANSFER Cause: CAUSE_UNKNOWN GC1: CallCtlTermConnDroppedEv TERMB REAASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: ConnDisconnectedEv A REASON=REASON_TRANSFER GC1: CallCtlConnDisconnectedEv A REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC1: CallInvalidEv REASON=REASON_TRANSFER GC2: ConnDisconnectedEv C REASON=REASON_TRANSFER GC2: CallCtlConnDisconnectedEv C REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: TermConnDroppedEv TERMB REASON=REASON_TRANSFER GC2: CallCtlTermConnDroppedEv TERMB REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: ConnDisconnectedEv B REASON=REASON_TRANSFER GC2: CallCtlConnDisconnectedEv B REASON=REASON_TRANSFER CallControlCause: CAUSE_TRANSFER GC2: CallInvalidEv REASON=REASON_TRANSFER GC2: CallObservationEndedEv REASON=NORMAL Cause: CAUSE_NORMAL GC1 CiscoTransferEndEv REASON=REASON_TRANSFER Cause: CAUSE_NORMAL GC2 CallObservationEndedEv REASON=NORMAL Cause: CAUSE_NORMAL

シナリオ 2

A が B にコールします。call=GC1。B はコールを 99999 にパークします。C はコール GC2 を使用してコールをパーク解除します。

操作	イベント
コールがパークされているときに、A のコール オブザーバに配信されるイベント	GC1: ConnDisconnectedEv B REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON=REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC1: ConnCreatedEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: ConnInProgressEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL GC1: CallCtlConnQueuedEv 9999 REASON=REASON_PARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK
GC2 を使用してコールがパーク解除されたとき	GC2: CiscoCallChangedEv Surviving= GC2 origcall= GC1 address= A REASON=REASON_UNPARK CallActiveEv REASON=REASON_UNPARK Cause: CAUSE_NEW_CALL GC2: ConnCreatedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: ConnConnectedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlConnEstablishedEv A REASON=REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK GC2: TermConnCreatedEv TERMA REASON=REASON_UNPARK GC2: TermConnActiveEv TERMA REASON=REASON_UNPARK Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TERMA REASON=REASON_UNPARK Cause: CAUSE_NORMAL CallControlCause: CAUSE_PARK

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

シナリオ 3

A が B にコールし、B は C に対して応答なしコール転送を実行します。B は応答せず、コールが C に提供されます。

操作	イベント

A のコール オブザーバに配信されるイベント	GC1: CallActiveEv REASON=NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnInitiatedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMA REASON=NORMAL GC1: TermConnActiveEv TERMA REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnTalkingEv TERMA REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnDialingEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

A のコール オブザーバに配信されるイベント （続き）	GC1: ConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv C REASON=REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: ConnInProgressEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnAlertingEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnDisconnectedEv B REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON= REASON_FORWARDNOANSWER Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnConnectedEv C REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

シナリオ 4

A が B にコールし、B がそのコールを C にリダイレクトします。

操作	イベント

B のコール オブザーバに配信されるイベント	GC1: CallActiveEv REASON=NORMAL Cause: CAUSE_NEW_CALL GC1: ConnCreatedEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnInProgressEv REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnCreatedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: ConnConnectedEv A REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: ConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL GC1: TermConnCreatedEv TERMB REASON=NORMAL Cause: Other: 0 GC1: TermConnRingingEv TERMB REASON=NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEvImpl TERMB REASON=NORMAL Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL

B のコール オブザーバに配信されるイベント （続き）	GC1: ConnDisconnectedEv A REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv A REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: TermConnDroppedEv TERMB REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlTermConnDroppedEv TERMB REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: ConnDisconnectedEv B REASON=REDIRECT Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv B REASON=REDIRECT Cause: CAUSE_NORMAL CallControlCause: CAUSE_REDIRECTED GC1: CallInvalidEv REASON=REDIRECT Cause: CAUSE_NORMAL

半二重メディア

A および B における RTP イベント。

操作	RTP イベント	確認インターフェイス
A が B にコールし、B がコールに応答する。	A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す
	B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す
B がコールを保留にする。	A – CiscoRTPInputStoppedEv CiscoRTPOutputStoppedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す
	B – CiscoRTPInputStoppredEv CiscoRTPOutputStoppedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す
	A-CiscoRTPInputStartedEv	Ev.isHalfDuplex() が True を返す
B がコールを取得する。	A- CiscoRTPInputStoppredEv	Ev.isHalfDuplex() が True を返す
	A – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す
	B – CiscoRTPInputStartedEv CiscoRTPOutputStartedEv	Ev.isHalfDuplex() が false を返し Ev.isHalfDuplex() が false を返す

録音と監視

A および TA は、監視先または録音元のアドレスおよび端末です。

B および TB は、監視元のアドレスおよび端末です。

シナリオ 1

アプリケーション ユーザには録音機能だけが設定されています。

操作	イベント	コール情報
ciscoProvider.getCapabilities().canRecord()	JTAPI が TRUE を返す	NA:
ciscoProvider.getCapabilities().canMonitor()	JTAPI が FALSE を返す	NA:

シナリオ 2

管理者により、ユーザの監視機能が有効化されています。

操作	イベント	コール情報
	CiscoProviderCapabilityChangedEv このイベントの hasMonitorCapabilityChanged() が true を返す。 hasRecordingCapabilityChanged() が true を返す。	NA
ciscoProvider.getCapabilities().canMonitor()	JTAPI が true を返す。	NA

シナリオ 3

録音の設定。A には、自動録音が設定されています。

操作	イベント	コール情報
ciscoAddressA.getRecordingConfig()	CiscoAddress.AUTO_RECORDING を返す。	NA:
アドレス上の録音ステータスは、アプリケーションによって制御される録音に変更される。	CiscoAddressRecordingConfigChangedEv	NA
CiscoAddressRecordingConfigChangedEv.getRecordingConfig()	CiscoAddress.APPLICATION_CONTROLLED	
ciscoAddressA.getRecordingConfig()	CiscoAddress.APPLICATION_CONTROLLED	

シナリオ 4

自動録音。A には、自動録音が設定されています。発信側 X が A にコールします。A が、コールに応答します。A にはコール オブザーバがあります。TA にはオブザーバがあります。

操作	イベント	コール情報
A が切断する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **CallCtlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CiscoRTPOutputStartedEv **CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPInputStartedEv **CiscoTermConnRecordingTargetInfoEv** **CiscoTermConnRecordingEndEv** Cause: CAUSE_NORMAL **CallCtlTermConnDroppedEv** TA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CallInvalidEv	Calling: X Called: A LRP: null Current calling: X Current called: A

シナリオ 5

自動録音。A には、自動録音が設定されています。発信側 X が A にコールします。A が、コールに応答します。アプリケーションが、startRecording() をコールします。

操作	イベント	コール情報
Call.startRecording()	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **CallCtlTermConnTalkingE**v TA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL JTAPI が例外をスローする。 CiscoRTPOutputStartedEv **CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPInputStartedEv CiscoTermConnRecordingTargetInfoEv	Calling: X Called: A LRP: null Current calling: X Current called: A

シナリオ 6

自動録音。A には、自動録音が設定されています。発信側 X が A にコールします。A が、コールに応答します。アプリケーションが、startRecording() をコールします。

操作	イベント	コール情報
Call.startRecording()	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **CallCtlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL JTAPI が例外をスローする。 CiscoRTPOutputStartedEv **CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPInputStartedEv CiscoTermConnRecordingTargetInfoEv	Calling: X Called: A LRP: null Current calling: X Current called: A

シナリオ 7

StartRecording()。A の録音設定は、アプリケーションにより制御されています。発信側 X が A にコールします。アプリケーションが、startRecording() をコールします。A が、コールに応答します。アプリケーションが、startRecording() をコールします。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL ... **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL	Calling: X Called: A LRP: null Current calling: X Current called: A
Call.startRecording()	JTAPI が、InValidStateException をスローする。	
A が、コールに応答する。	CiscoRTPOutputStartedEv **CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPInputStartedEv	
Call.startRecording()	**CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL **CiscoTermConnRecordingTargetInfoEv**	

シナリオ 8

stopRecording()。A の録音設定は、アプリケーションにより制御されています。発信側 X が A にコールします。A が、コールに応答します。アプリケーションが、stopRecording() をコールします。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL ... **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL	Calling: X Called: A LRP: null Current calling: X Current called: A
A が、コールに応答する。	**CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	
Call.startRecording()	**CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL **CiscoTermConnRecordingTargetInfoEv**	
Call.stopRecording()	**CiscoTermConnRecordingEndEv** Cause: CAUSE_NORMAL	
A が切断する。	**CallCtlTermConnDroppedEv** TA Cause: CAUSE_NORMAL CallControlCause: CAUSE_NORMAL CallInvalidEv	

シナリオ 9

保留。A には、自動録音が設定されています。発信側 X が A にコールします。A が、コールに応答し、コールを保留します。A が、コールを再開します。RTP イベントが端末オブザーバに配信され、それらはコール イベントに依存しません。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL	Calling: X
	ConnCreatedEv for A Cause: CAUSE_NORMAL	Called: A
	ConnConnectedEv for A Cause: CAUSE_NORMAL	LRP: null
	...	Current calling: X
	CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL	
	CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	Current called: A
A が、コールに応答する。	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	
	CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL	
	CiscoTermConnRecordingTargetInfoEv	
	CiscoRTPOutputStoppedEv	
A が、コールを保留にする。	CallCrlTermConnHeldEv Cause: CAUSE_NORMAL	
	CiscoRTPInputStoppedEv	
	CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL	
	...	
	...	
	CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	
A が、コールを再開する。	**CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL	
	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	
	CiscoTermConnRecordingTargetInfoEv	
	CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL	
	CallCtlTermConnDroppedEv TA Cause: CAUSE_NORMAL	
	CallControlCause: CAUSE_NORMAL	
	CiscoRTPOutputStoppedEv	
A が、コールを破棄する。	CallInvalidEv	
	CiscoRTPInputStoppedEv	

シナリオ 10

会議コントローラおよび録音元。A には、自動録音が設定されています。発信側 X が A にコールします。A が、コールに応答します。A が Y にコンサルト コールを開始します。Y が応答し、A が会議を開催します。

操作	イベント	コール情報
A が、コール GC1 に応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause: CAUSE_NORMAL ... **CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPOutputStoppedEv **CiscoTermConnRecordingTargetInfoEv**	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
A が GC2 で Y にコンサルト コールする。	**GC1:CallCrlTermConnHeldEv** Cause: CAUSE_NORMAL CiscoRTPInputStoppedEv **GC1:CiscoTermConnRecordingEndEv** Cause: CAUSE_NORMAL ... ... **CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **GC2:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC2:ConnConnectedEv** for A Cause: CAUSE_NORMAL	

操作	イベント	コール情報
A が会議を開催する。	**GC2:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL GC2: ConnConnected Y CiscoRTPOutputStartedEv **GC2:CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPInputStartedEv **CiscoTermConnRecordingTargetInfoEv** CiscoConferenceStartEv(GC2 -> GC1) **GC2: CiscoTermConnRecordingEndEv** TA **GC2:CallCtlTermConnDroppedEv** TA Cause: CAUSE_NORMAL GC2:CiscoRTPOutputStoppedEv ... GC2:CallInvalidEv GC2:CiscoRTPInputStoppedEv GC1: CallCtlTermConnTalkingEv TA GC1: ConnCreatedEv X GC1: ConnCreatedEv Y ... **GC1: CiscoTermConnRecordingStartEv** TA CiscoConferenceEndEv **CiscoTermConnRecordingTargetInfoEv** CiscoRTPOutputStartedEv CiscoRTPInputStartedEv （録音の開始は、会議終了イベント後に表示される場合がある）	

シナリオ 11

会議の対象および録音元。A には、自動録音が設定されています。発信側 X が Y GC1 にコールします。Y が A にコンサルト コールして、A がコール（GC2）に応答し、Y が会議を開催します。

操作	イベント	コール情報
A が、コール GC2 に応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **GC2:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC2:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC2: ConnConnectedEv Y ... **GC2:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC2:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL **CiscoTermConnRecordingTargetInfoEv**	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
Y が会議を開催する。	GC1: CallActiveEv **GC1: ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1: ConnCreatedEv** for Y Cause: CAUSE_NORMAL **CiscoConferenceStartEv**(GC2->GC1) CiscoCallChangedEv ... CiscoRTPOutputStoppedEv CiscoRTPInputStoppedEv GC2: CallCrlTermConnDroppedEv TA GC2: CallInvalidEv ... CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC1: CallCrlTermConnTalkingEv** TA GC1: ConnConnectedEv Y GC1: ConnConnectedEv X GC1: CiscoConferenceEndEv ... （録音の開始は、会議終了イベント前に表示される場合がある）	

シナリオ 12

転送コントローラおよび録音元。A には、録音が設定されています。発信側 X が A にコールします。A が、コールに応答します。A が Y にコンサルト コールを開始します。Y が応答し、A による転送が成立します。

操作	イベント	コール情報
A が、コール GC1 に応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL	GC1:
	ConnCreatedEv for A Cause: CAUSE_NORMAL	Calling: X
	ConnConnectedEv for A Cause: CAUSE_NORMAL	Called: A
	...	LRP: null
	CallCrlTermConnRingingEv TA Cause: CAUSE_NORMAL	Current calling: X
	CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	Current called: A
	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	
	CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL	
	CiscoTermConnRecordingTargetInfoEv	
A が GC2 で Y にコンサルト コールする。		
	CiscoRTPOutputStoppedEv	
	GC1:CallCrlTermConnHeldEv Cause: CAUSE_NORMAL	
	CiscoRTPInputStoppedEv	
	GC1:CiscoTermConnRecordingEndEv Cause: CAUSE_NORMAL	
	...	
	...	GC2:
	CallActiveEv for callID=GC2 Cause: CAUSE_NEW_CALL	Calling: A
	GC2:ConnCreatedEv for A Cause: CAUSE_NORMAL	Called: Y
	GC2:ConnConnectedEv for A Cause: CAUSE_NORMAL	LRP: null
	...	Current calling: A
	...	Current called: Y

操作	イベント	コール情報
A による転送を実行する。	**GC2:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL GC2: ConnConnected Y CiscoRTPOutputStartedEv **GC2:CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL CiscoRTPInputStartedEv **CiscoTermConnRecordingTargetInfoEv** CiscoTransferStartEv(GC2 -> GC1) GC2:CiscoTermConnRecordingEndEv **GC2:CallCtlTermConnDroppedEv** TA GC2:CiscoRTPOutputStoppedEv ... GC2:CallInvalidEv GC2:CiscoRTPInputStoppedEv **GC1: CallCtlTermConnDroppedEv** TA ... GC1: CallInvalidEv CiscoTransferEndEv CiscoRTPOutputStartedEv CiscoRTPInputStartedEv （録音の終了は、A による転送成立後に表示されない場合がある）	GC1: Calling: X Called: Y LRP: A Current calling: X Current called: Y

シナリオ 13

転送先および録音元。A には、自動録音が設定されています。発信側 X が Y GC1 にコールします。Y が A にコンサルト コールして、A がコール（GC2）に応答し、Y による転送を実行します。

操作	イベント	コール情報
A が、コール GC2 に応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **GC2:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC2:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC2: ConnConnectedEv Y ... **GC2:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC2:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv **GC2:CiscoTermConnRecordingStartEv** Cause: CAUSE_NORMAL **CiscoTermConnRecordingTargetInfoEv**	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
Y が転送を実行する。	GC1: CallActiveEv **GC1: ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1: ConnCreatedEv** for Y Cause: CAUSE_NORMAL **CiscoTransferStartEv**(GC2->GC1) CiscoCallChangedEv ...	
発信側または A が、コールを破棄する。	CiscoRTPOutputStoppedEv CiscoRTPInputStoppedEv **GC1: CallCrlTermConnTalkingEv** TA **GC1: CallCtlConnDisconnectedEv** Y **GC1: ConnConnectedEv** X **GC2: CallCrlTermConnDroppedEv** TA GC2: CallInvalidEv ... CiscoRTPOutputStartedEv CiscoRTPInputStartedEv ... **GC1:CiscoTermConnRecordingEndEv** ... GC1: CallInvalidEv	

シナリオ 14

次の監視を開始および停止します。A は監視対象、B は監視元です。X が A にコールし、A がコール GC1（ci1）に応答します。B が、GC2 を使用して、監視の開始をコールします。アプリケーションには、A と B の両方にコール オブザーバがあります。アプリケーションには、監視機能が有効化されています。

操作	イベント	コール情報
A が、GC1 に応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
B が、GC2 を使用して、 監視の開始をコールし、	**GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A	
CI1、A、および TermA を GC1 から指定する。	（A または GC2 に端末接続なし） GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, terminal name: TA, rtphandle=CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorInitiatorInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	GC2: Calling: B Called: A LRP: null Current calling: B Current called: A

操作	イベント	コール情報
A が、コールを保留にする。	GC2: CiscoRTPOutputStoppedEv GC1: CiscoRTPOutputStoppedEv **GC1: CallCtlTermConnHeldEv** TA **GC2**:CiscoRTPInputStoppedEv **GC1**: CiscoRTPInputStoppedEv	
A が、コールを再開する。	**GC1**: CiscoRTPOutputStartedEv GC2: CiscoRTPOutputStartedEv GC1: CallCtlTermConnTalking TA **GC2**:CiscoRTPInputStartedEv **GC1:** CiscoRTPInputStartedEv	
B が、GC2 の破棄をコールして、監視を停止する。	GC2: CallCtlTermConnDroppedEv TB GC2: ConnDisconnEv A GC1: CiscoTermConnMonitorEndEv TA GC2: ConnDisconnEv B GC2: CallInvalidEv	

シナリオ 15

監視元が、Y にコールを転送します。A は監視対象、B は監視元です。X が A にコールし、A がコール GC1（ci1）に応答します。B が、GC2 を使用して、監視の開始をコールします。アプリケーションには、A と B の両方にコール オブザーバがあります。アプリケーションには、監視機能が有効化されています。B が、Y にコールを転送します。

操作	イベント	コール情報
A が、GC1 に応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
B が、GC2 を使用して、 監視の開始をコールし、 CI1、A、および TermA を GC1 から指定する。 または、A の端末接続を使用。	**GC2:CallActive** Cause: CAUSE_NORMAL **GC2: ConnConnectedE**v for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A （A または GC2 に端末接続なし） GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL Monitor_TARGET address:A, device name: TA, rtphandle=CI1 **GC1: CiscoTermConnMonitorStartEv TA** **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_NORMAL address:B, device name: TB rtphandle=ci2	

操作	イベント	コール情報
B が、GC3 を使用して Y にコンサルト コールし、転送を実行する。	GC3**:** CallActiveEv GC3: ConnConnectedEv B GC3: CallCtlTermConnTalkingEv TB GC3: ConnConnectedEv Y CiscoTransferStartEv(GC3->GC2) GC3: ConnDisconnectedEv Y GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_NORMAL address:Y, device name: TY GC3: ConnDisconnectedEv B GC3: CallCtlTermConnDroppedEv TB GC3: CallInvalidEv GC2: CallCtlTermConnDroppedEv TB ... GC2: CallInvalidEv CiscoTransferEndEv	GC1: Calling: X Called: A LRP: A Current calling: X Current called: Y
Y のコール オブザーバの場合	GC3: CallActive **GC3:** CallCtlTermConnRingingEv TY GC3: CallCtlTermConnTalkingEv TY CiscoTransferStartEv CiscoCallChangedEv GC3->GC2 GC2: ConnConnectedEv Y **GC2**: ConnConnectedEv B GC2: CiscoTermConnMonitorTargetInfoEv TY Cause: CAUSE_NORMAL address:A, device name: TA GC3: ConnDisconnectedEv Y **GC3:** CallCtlTermConnDroppedEv TY GC3: CallInvalidEV GC2: ConnConnectedEv X GC2: ConnDisconnectedEv A CiscoTransferEndEv （GC1 の **CiscoTermConnMonitorInitiatorInfoEv** は、GC3 および GC2 の転送イベントに依存せず、終了イベントの前や後に随時送信できる。）	

シナリオ 16

割り込みされたコールを監視します。A および A’ は、共用回線です。送信者が A にコールし、A がコールに応答します。A’ が、コールに割り込みします。B が、監視の開始をコールします。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL	GC1:
	GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL	Calling: X
	GC1:ConnConnectedEv for A Cause: CAUSE_NORMAL	Called: A
	GC1: ConnConnectedEv X	LRP: null
	...	Current calling: X
A が、GC1 に応答する。	**GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL	Current called: A
	GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	
	CiscoRTPOutputStartedEv	
	CiscoRTPInputStartedEv	
	GC1: CallCtlTermConnBridgedEv TermA'	
A' はコールに割込む。	GC1: **GC1:CallCrlTermConnTalkingEv** TA'	
	startMonitor 要求に例外がスローされる。	
B が、GC2 を使用して、 監視の開始をコールし、 CI1、A、および TermA を GC1 から指定する。		

シナリオ 17

監視および録音。A は、監視対象で、自動録音が設定されています。B は、監視元です。X が A にコールし、A がコール GC1（ci1）に応答します。B が、GC2 を使用して、監視の開始をコールします。アプリケーションには、A と B の両方にコール オブザーバがあります。アプリケーションには、監視機能が有効化されています。

操作	イベント	コール情報
A が、GC1 に応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnRingingEv** TA Cause: CAUSE_NORMAL **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL CiscoRTPOutputStartedEv GC1: CiscoTermConnRecordingStartEv TA **GC1: CiscoTermConnRecordingTargetInfoEv TA** CiscoRTPInputStartedEv	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
B が、GC2 を使用して、 監視の開始をコールし、 CI1、A、および TermA を GC1 から指定する。	**GC2:CallActive** Cause: CAUSE_NORMAL **GC2: GC1:ConnConnectedEv** for B Cause: CAUSE_NORMAL GC2: CallCtlTermConnTalkingEv TB GC2: ConnCreatedEv A （GC2 上の A に端末接続なし） GC2: ConnConnectedEv A **GC2:CiscoTermConnMonitorTargetInfoEv** Cause: CAUSE_NORMAL address:A, device name: TA, rtphandle=CI1 **GC1: CiscoTermConnMonitorStartEv** TA **GC1: CiscoTermConnMonitorTargetInfoEv** TA Cause: CAUSE_NORMAL address:B, device name: TB	

シナリオ 18

モニタリングで、Remote in Use の共用回線を次のように監視しています。A および A’ は、共用回線です。送信者が A にコールし、A がコールに応答します。B が、監視の開始をコールします。アプリケーションには、A’ だけにコール オブザーバがあります。B は、A への接続済みコールに監視要求を開始します。A’ のコール オブザーバには、開始イベントは配信されません。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL **GC1:ConnCreatedEv** for A' Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A' Cause: CAUSE_NORMAL GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnRingingEv** TA' Cause: CAUSE_NORMAL GC1: CallCtlTermConnBridgedEv TermA' Cause: CAUSE_NORMAL	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
A が GC1 に応答し、B が監視を開始する。	**GC1: CallCrlTermConnHeldEv TA'**	
A が、コールを保留にする。	**GC1:CallCrlTermConnTalkingEv TA'**	
A' はコールに応答する。	GC1: CiscoTermConnMonitorStartEv TA'	
B が、コールを破棄し、GC3 を使用して監視を開始し、監視要求で A’ の端末接続を指定する。	**GC1: CiscoTermConnMonitorInitiatorInfoEv TA'** Cause: CAUSE_NORMAL address:B, device name: TB	

シナリオ 19

監視および録音のためのスナップ ショット イベント。A は、監視対象で、自動録音が設定されています。B は、監視元です。X が A にコールし、A がコール GC1（ci1）に応答し、B が GC2 を使用して監視の開始をコールします。監視および録音セッションの確立後、別のアプリケーションにより、A にコール オブザーバが追加されます。

操作	イベント	コール情報
	CallActiveEv for callID=GC1 Cause: CAUSE_SNAPSHOT	GC1:
	GC1:ConnCreatedEv for A Cause: CAUSE_SNAPSHOT	Calling: X
	GC1:ConnConnectedEv for A Cause: CAUSE_SNAPSHOT	Called: A
	GC1: ConnConnectedEv X	LRP: null
	...	Current calling: X
	GC1:CallCrlTermConnTalkingEv TA Cause: CAUSE_SNAPSHOT	Current called: A
	GC1: CiscoTermConnRecordingStartEv TA Cause: CAUSE_SNAPSHOT	
	GC1: CiscoTermConnRecordingTargetInfoEv TA Cause: CAUSE_SNAPSHOT	
	GC1: CiscoTermConnMonitorStartEv TA Cause: CAUSE_SNAPSHOT	
	GC1: CiscoTermConnMonitorInitiatorInfoEv TA Cause: CAUSE_SNAPSHOT address:B, device name: TB	

シナリオ 20

チェーニングされた会議および録音。A には、自動録音が設定されています。A、B、および C が GC1 で会議中です。A が GC3 を使用して D にコンサルト コールします。D は、Gc4 を使用して、（E および F との）会議を設定し、A のコンサルト コールを会議に追加します。A は会議を開催し、会議はチェーニングされます。

操作	イベント	コール情報
B が A にコールし、A が GC1 でコールに応答する。	**CallActiveEv** for callID=GC1 **GC1:ConnCreatedEv** for A Cause: CAUSE_NORMAL **GC1:ConnConnectedEv** for A Cause: CAUSE_NORMAL GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_NORMAL **GC1: CiscoTermConnRecordingStartEv TA** Cause: CAUSE_NORMAL **GC1: CiscoTermConnRecordingTargetInfoEv TA** Cause: CAUSE_NORMAL **GC1: CiscoTermConnMonitorStartEv TA** Cause: CAUSE_NORMAL **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_NORMAL address:B, device name: TB	NA
A が GC2 を使用して C にコンサルト コールする。	GC1: TermConnHeldEv TA GC2: CallActiveEv GC1: **CiscoTermConnRecordingEndEv TA** Cause: CAUSE_NORMAL ... GC2: **CallCrlTermConnTalkingEv TA** Cause: CAUSE_NORMAL GC2: **CiscoTermConnRecordingStartEv TA** Cause: CAUSE_NORMAL ...	

操作	イベント	コール情報
A が会議を開催する。	GC2: **CiscoTermConnRecordingEndEv TA** Cause: CAUSE_NORMAL ..... GC2: CallInvalidEv GC1: **CiscoTermConnRecordingStartEv TA** Cause: CAUSE_NORMAL GC1: TermConnTalkingEv TA	
A が GC3 を使用して D にコンサルト コールする。	GC3: CallActiveEv GC1: CallCrlTermConnHeldEv TA **GC1: CiscoTermConnRecordingEndEv TA** Cause: CAUSE_NORMAL	
D が応答する。	GC3: CallCrlTermConnTalkingEv TA GC3: **CiscoTermConnRecordingStartEv TA** Cause: CAUSE_NORMAL	
D は、E および F とコンサルト コールし、会議を開催する。	GC3: CiscoTermConnRecordingEndEv TA GC1: CallCrlTermConnTalkingEv TA ...	
A が会議を開催する。	GC3: CallInvalidEv GC1: CiscoTermConnRecordingStartEv	

インターコム

設定：端末 T1 には、TargetDN が B、ラベルが Bob、Unicode ラベルが UBob のインターコム回線 A があります。端末 T2 には、インターコム回線 B があります。アプリケーション プロバイダーでは、コントロール リストに T1 と T2 の両方があります。

C、Carol、UCarol は、A および B と同じインターコム グループに属しています。

D、David、UDavid は、A、B、および C と同じインターコム グループに属していません。

操作	結果	コール情報
アプリケーションがプロバイダーをオープンし、プロバイダーがイン サービスになると、アプリケーションは provider.getIntercomAddresses() を発行する。	JTAPI は、CiscoIntercomAddress の配列として、A および B を返す。	N.A.
アプリケーションは、CiscoIntercomAddress.getIntercomTargetDN()、CiscoIntercomAddress.getIntercomTargetLabel()、および CiscoIntercomAddress.getIntercomUnicodeTarget Label() 要求を A で発行する。	JTAPI は、TargetDN として B を、TargetLabel として Bob および UBob を返す。	N.A.
アプリケーションは、CiscoIntercomAddress.getDafaultIntercomTargetDN()、CiscoIntercomAddress.getDefaultIntercomTargetLabel()、および CiscoIntercomAddress.getDefaultIntercomUnicodeTargetLabel() 要求を A で発行する。	JTAPI は、TargetDN として B を、TargetLabel として Bob および UBob を返す。	N.A.
アプリケーションは、インターコム アドレス A で、CiscoIntercomAddres.setIntercomTarget(C, Carol, UCarol) を発行する。 応答に成功したら、アプリケーションは、CiscoIntercomAddress.getIntercomTargetDN()、CiscoIntercomAddress.getIntercomTargetLabel()、および CiscoIntercomAddress.getIntercomUnicodeTargetLabel() 要求を A で発行する。	AddressObserver at A: CiscoAddrIntercomInfoChanged Ev Cause: CAUSE_NORMAL JTAPI は、TargetDN として C を、TargetLabel として Carol および UCarol を返す。	N.A.
Application1 は、CiscoIntercomAddress A を監視し、これに対して AddressObserverAdded を持っている。Application2 は、インターコム ターゲット、ラベルを、C、Carol、UCarol に設定する。	App1 : AddressObserver at A: CiscoAddrIntercomInfoChanged Ev Cause: CAUSE_NORMAL	N.A.
上記の手順の後、Application1 は、インターコム アドレス A で CiscoIntercomAddres.setIntercomTarget(B, Bob, UBob) を発行する。	別のアプリケーション インスタンスがすでにターゲットを C、Carol、UCarol に設定しているため、アプリケーションに例外がスローされる。	N.A
インターコム ターゲット DN およびインターコム アドレス A のラベルがデフォルトに設定されているため、アプリケーションは、インターコム アドレス A で CiscoIntercomAddres.setIntercomTarget(D, David, UDavid) を発行する。	D、David、UDavid が同じインターコム グループに属していないため、例外がスローされる。	N.A.

操作	結果	コール情報
アプリケーションは、インターコム アドレス A でインターコム ターゲット DN およびラベルを C、Carol、UCarol に設定している。ここで、CTIManager がアウト オブ サービスになり、JTAPI は別の CTIManager ノードにフェールオーバーする。インターコム アドレス A がイン サービスに戻った後、JTAPI は、インターコム ターゲット DN およびラベルを、それぞれ、C、Carol、UCarol に復旧する。 アプリケーションは、CiscoIntercomAddress.getIntercomTargetDN()、CiscoIntercomAddress.getIntercomTargetLabel()、および CiscoIntercomAddress.getIntercomUnicodeTargetLabel() 要求を A で発行する。	AddressObserver at A: CiscoAddrIntercomInfoChanged Ev Cause: CAUSE_NORMAL JTAPI は、TargetDN として C を、TargetLabel として Carol および UCarol を返す。	N.A.
アプリケーションは、インターコム アドレス A でインターコム ターゲット DN およびラベルを C、Carol に設定している。ここで、CTIManager がアウト オブ サービスになり、JTAPI は別の CTIManager ノードにフェールオーバーする。インターコム アドレス A がイン サービスに戻った後、JTAPI は、インターコム ターゲット DN、ラベルおよびユニコードラベルを、それぞれ、C、Carol、UCarol に復旧する。ただし、他のアプリケーションがすでにターゲット DN を設定しているという競合状態が原因で、JTAPI は CTI から失敗の応答を受信する。	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A.
アプリケーションは CTIManager ノードに接続されている。Cisco Unified Communications Manager ノードが停止し、インターコム アドレスがイン サービスに戻った後、インターコム デバイスは、別の Cisco Unified Communications Manager ノードにフェールオーバーする。CTIManager は、インターコム ターゲット DN およびラベルを復旧する必要がある。 アプリケーションは、CiscoIntercomAddress.getIntercomTargetDN()、CiscoIntercomAddress.getIntercomLabel()、および CiscoIntercomAddress.getIntercomUnicodeTargetLabel() 要求を A で発行する。	A の AddressObserver : CiscoAddrIntercomInfoChanged Ev Cause: CAUSE_NORMAL JTAPI は、TargetDN として C を、TargetLabel として Carol および UCarol を返す。	N.A.

操作	結果	コール情報
アプリケーションは CTIManager ノードに接続されている。Cisco Unified Communications Manager ノードが停止し、インターコム アドレスがイン サービスに戻った後、インターコム デバイスは、別の Cisco Unified Communications Manager ノードにフェールオーバーする。CTIManager は、インターコム ターゲット DN およびラベルの復旧を試みるが、他のアプリケーションがすでにターゲット DN を設定しているという競合状態が原因で、CTI はインターコム ターゲット DN およびラベルを復旧できない。	AddressObserver at A: CiscoAddrIntercomInfoRestorationFailedEv Cause: CAUSE_NORMAL	N.A.

操作	結果	コール情報
アプリケーションは、インターコム アドレス A および B を監視している。A は、B に設定されたターゲットを持つ。ユーザが、インターコム コールを開始する。 インターコム コールは成功する。	A および B の CallObserver : CallActiveEv GC1 Cause: CAUSE_NORMALConnCreatedEv A, Cause: CAUSE_NORMALConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMALTermConnCreatedEv A- T1 Cause: CAUSE_NORMAL TermConnActiveEv A- T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv A - T1 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL ConnCreatedEv B, Cause: CAUSE_NORMAL ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL	Cg=A Cd=B CurrentCg=A CurredCd=B LRP = null

操作	結果	コール情報
	TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtl Cause=CAUSE_NORMAL CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	
B のユーザは、talkback ソフトキーを押して、インターコムの発信者に接続する。	A および B の CallObserver： TermConnActiveEv B - T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CiscoRTPOutputStartedEv – T2CiscoRTPInputStartedEv – T1	Cg=A Cd=B CurrentCg=A CurredCd=B LRP = null

操作	結果	コール情報
インターコム アドレス A は、B に定義されたターゲットを持っている。アプリケーションは、空の電話番号で、インターフェイス Address.ConnectIntercom() をコールして、インターコム コールを開始する。 インターコム コールは成功する。	A および B の CallObserver : CallActiveEv GC1 Cause: CAUSE_NORMAL ConnCreatedEv A Cause: CAUSE_NORMAL ConnConnectedEv A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL TermConnCreatedEv T1 Cause: CAUSE_NORMAL TermConnActiveEv T1 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv T1 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnDialingEv A Cause: CAUSE_NORMAL CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL ConnCreatedEv B Cause: CAUSE_NORMAL C ConnConnectedEv B Cause: CAUSE_NORMAL CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL TermConnCreatedEv B- T2 Cause: CAUSE_NORMAL TermConnPassiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnBridgeEv B – T2 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL	Cg=A Cd=B CurrentCg=A CurredCd=B LRP = null

操作	結果	コール情報
	CiscoToneChangedEv – T1 –GC1 CiscoToneChangedEv – T2 –GC1 CiscoRTPOutputStartedEv – T1 CiscoRTPInputStartedEv – T2	
アプリケーションは、talkback のために、B の TerminalConnection の TerminalConnection.join() 要求を開始する。 要求が成功する。	A および B の CallObserver : TermConnActiveEv B – T2 Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv B – T2 Cause: CAUSE_NORMAL CallCtlCause=CAUSE_NORMAL CiscoRTPOutputStartedEv – T2 CiscoRTPInputStartedEv – T1	Cg=A Cd=B CurrentCg=A CurredCd=B LRP = null
アプリケーションは、TerminalConnection.hold() を発行して、A でのインターコム コールの保留を試みる。	PlatformException がスローされ、インターコム コールは接続を維持。	N.A.
アプリケーションは、B の接続で connection.accept() を発行して、インターコム ターゲットでのインターコム コールの受信を試みる。	PlatformException がスローされ、インターコム コールは接続を維持。	N.A.
アプリケーションは、B の接続で connection.reject() を発行して、インターコム ターゲットでのインターコム コールの拒否を試みる。	インターコム コールは、接続解除される。	N.A.
アプリケーションは、A または B の接続で connection.redirect() を発行して、インターコム コールのリダイレクトを試みる。	PlatformException がスローされ、インターコム コールは接続を維持。	N.A.
アプリケーションは、A または B の接続で connection.park() を発行して、コールのパークを試みる。	PlatformException がスローされ、インターコム コールは接続を維持。	N.A.
端末 T1 は、インターコム ターゲットが B に設定されたインターコム アドレス A を持っている。 端末 T2 は、インターコム アドレス B および他のアドレス C を持っている。C は、D と GC1 でコール中。 A は B に対してインターコム コールを開始し、インターコム コールは B で自動応答される。	GC1 コールへのイベントなし。Connected State のまま。	N.A.
アプリケーションは、CiscoIntercomAddress. setForwarding () を発行して、インターコム アドレス A で setForwarding を試みる。	PlatformException がスローされる。	N.A.

操作	結果	コール情報
アプリケーションは、CiscoIntercomAddress.setRingerStatus() を発行して、インターコム アドレス A で setRingerStatus を試みる。	PlatformException がスローされる。	N.A.
アプリケーションは、CiscoIntercomAddress.setMessageWaiting() を発行して、インターコム アドレス A で setMessageWaiting を試みる。	PlatformException がスローされる。	N.A.
アプリケーションは、CiscoIntercomAddress.setAutoAcceptStatus () を発行して、インターコム アドレス A で setAutoAcceptEnabled を試みる。	PlatformException がスローされる。	N.A.
アプリケーションは、CiscoIntercomAddress.getAutoAcceptStatus () を発行して、CTIPort のインターコム アドレス A で getAutoAcceptEnabled を試みる。	PlatformException がスローされる。	N.A.

DeviceState Whisper のシナリオ

設定：端末 T1 はインターコム アドレス B を持ち、端末 T2 はインターコム アドレス A を持っています。アプリケーションは、CiscoTermDeviceStateWhisperEv および他のすべての DeviceState Events が T1 および T2 上で有効となるように CiscoTermEvFilter を設定しています。アプリケーションは、T1 と T2 の両方に端末オブザーバを追加しています。

操作	イベント	コール情報
インターコム アドレス A は、B に定義されたターゲットを持っている。アプリケーションは、空の電話番号で、インターフェイス Address.ConnectIntercom() をコールして、インターコム コールを開始する。	T1 の TerminalObserver で受信されるイベント CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL T2 の TerminalObserver で受信されるイベント CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_NORMAL	N.A.
アプリケーションは、T2（インターコム ターゲット）の TerminalConnection で、T1（インターコム イニシエータ）への talkback のために join() 要求を発行する。	T1 の TerminalObserver で受信されるイベント なし。 T2 の TerminalObserver で受信されるイベント CiscoTermDeviceStateActiveEv T1 Cause: CAUSE_NORMAL	N.A.
端末 T2 にはすでにインターコム ターゲット コールがあり、アプリケーションは、CiscoTermDeviceStateWhisperEv に対する CiscoTermFilter を有効化する。	**T2 の TerminalObserver で受信されるイベント** CiscoTermDeviceStateWhisperEv T1 Cause: CAUSE_SNAPSHOT	N.A.

Do Not Disturb（サイレント）

設定：アプリケーションは、端末 A と端末 B を監視しています。

シナリオ 1

アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加します。フィルタは、setDNDChangedEvFilter によって有効化されます。DND（サイレント）は、端末上で有効化されます。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出します。

操作	イベント	コール情報
アプリケーションが、端末 A に端末オブザーバを追加する。フィルタは、CiscoTermEvFilter 内の setDNDChangedEvFilter() によって有効化される。DND（サイレント）は、phone または admin ページにより端末上で有効化される。 アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出す。	NEW META EVENT_____ META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true がアプリケーションに返される。	N.A.

シナリオ 2

アプリケーションが、イベントの受信用にフィルタを有効化します。アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加します。DND（サイレント）は、端末上で有効化されます。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出します。

操作	イベント	コール情報
アプリケーションが、イベントの受信用にフィルタを有効化する。アプリケーションが、端末 A に端末オブザーバを追加する 。DND（サイレント）は、phone または admin ページによりデバイス上で有効化される。 アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出す。	NEW META EVENT_____META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true がアプリケーションに返される。	N.A.

シナリオ 3

アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加します。フィルタは、CiscoTermEvFilter 内の setDNDChangedEvFilter() によって無効化されます。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出します。

操作	イベント	コール情報
アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加する。フィルタは、CiscoTermEvFilter 内の setDNDChangedEvFilter() によって無効化される。 アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出す。	CiscoTermDNDStatusChangedEv は、アプリケーションに配信されていない。	N.A.

シナリオ 4

アプリケーションは、端末に端末オブザーバを追加しません。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出します。

操作	イベント	コール情報
アプリケーションは、端末に端末オブザーバを追加しない。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出す。	InvalidStateException がスローされる。	N.A.

シナリオ 5

アプリケーションは、イベントの受信用にフィルタを有効化しません。アプリケーションが、端末 A に端末オブザーバを追加します。DND（サイレント）ステータスは、phone または admin ページにより true に設定されます。これにより、アプリケーションは、フィルタによるイベントの受信を有効化します。アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出します。

操作	イベント	コール情報
アプリケーションは、イベントの受信用にフィルタを有効化しない。アプリケーションが、端末 A に端末オブザーバを追加する。DND（サイレント）ステータスは、phone または admin ページにより true に設定される。 これにより、アプリケーションは、フィルタによるイベントの受信を有効化する。 アプリケーションが、CiscoTerminal から getDNDStatus() を呼び出す。	CiscoTermDNDStatusChangedEv は、アプリケーションに配信されていない。 NEW META EVENT_____META_CALL_STARTING CiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: true DND status = true がアプリケーションに返される。	N.A.

シナリオ 6

アプリケーションは、CiscoTerminal で setDNDStatus() インターフェイスを呼び出して、DND（サイレント）ステータスを false に設定します。

操作	イベント	コール情報
アプリケーションが、CiscoTerminal から setDNDStatus() を呼び出す。	NEW META EVENT_____META_CALL_STARTINGCiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A.

シナリオ 7

Application1 および Application2 は、端末 A を監視しており、両アプリケーションはイベントの受信用にフィルタを有効化しています。Application1 は、端末 A で、DND（サイレント）ステータスを false に設定します。Application2 は、端末 A を監視しています。

操作	イベント	コール情報
アプリケーションが、CiscoTerminal から setDNDStatus() を呼び出す。	NEW META EVENT_____META_CALL_STARTINGCiscoTermDNDStatusChangedEv A Cause: CAUSE_NORMAL for DND Status: false	N.A.

シナリオ 8

DND（サイレント）タイプは RingerOff で、CFNA は設定されていません。端末 B が端末 A をコールします。コールは A に表示されるが、コールは応答されません。

操作	イベント	コール情報
アプリケーションは、CiscoCall から、feature priority の設定 3 で redirect() API を呼び出す。	コールは、デバイスの DND（サイレント）設定に関わりなく、デバイスに表示される。CER コールは、DND（サイレント）設定よりも優先される。	N.A.
アプリケーションは、CiscoRouteSession から、feature priority の設定 3 で selectRoute() API を呼び出す。	コールは、デバイスの DND（サイレント）設定に関わりなく、デバイスに表示される。CER コールは、DND（サイレント）設定よりも優先される。	N.A.

シナリオ 9

操作	イベント	コール情報
DND（サイレント）タイプは RingerOff で、CFNA は設定されていない。端末 B が端末 A をコールする。コールは A に表示されるが、コールは応答されない。	ConnFailedEv Cause: CAUSE_NO ANSWER	N.A.

シナリオ 10

DND（サイレント）タイプは CallReject で、CFB は設定されていません。端末 B が端末 A をコールします。コールは A に表示されません。

操作	イベント	コール情報
DND（サイレント）タイプは CallReject で、CFB は設定されていない。端末 B が端末 A をコールする。コールは A に表示されない。	ConnFailedEv Cause: CAUSE_USER BUSY	N.A.

シナリオ 11

DND（サイレント）は、端末 A で有効化されます。端末 A は IN_SERVICE となります。アプリケーションが、ServiceEv の CiscoTerm で getDNDStatus() を呼び出します。

操作	イベント	コール情報
DND（サイレント）は、端末 A で有効化される。端末 A は IN_SERVICE となる。	CiscoTermInServiceEv Cause: CAUSE_NORMAL DND Status =true	N.A.

シナリオ 12

DND（サイレント）は、端末 A で有効化されます。端末 A は IN_SERVICE となります。アプリケーションが、setDNDStatus() を呼び出します。setDNDStatus() 要求が送信された後、DB 障害が発生します。

操作	イベント	コール情報
DND（サイレント）は、端末 A で有効化される。端末 A は IN_SERVICE となる。アプリケーションが、setDNDStatus() を呼び出す。DB 障害が発生し、値は DB 内で更新されない。	PlatformException で「Could not meet post conditions of setDNDStatus()」がスローされる。 CiscoTermDNDStatusChangedEv の受信なし。	N.A.

シナリオ 13

DND（サイレント）は、端末 A で有効化されます。端末 A は IN_SERVICE となり、現在、phone/admin の DND（サイレント）ステータスは true です。アプリケーションは同じ値の設定を試みます。つまり、setDNDStatus(true) を呼び出します。

操作	イベント	コール情報
DND（サイレント）は、端末 A で有効化される。端末 A は IN_SERVICE となり、現在、phone/admin の DND（サイレント）ステータスは true である。アプリケーションは、同じ値の設定を試みる。つまり setDNDStatus(true) を呼び出す。	次の InvalidStateException がキャッチされる。「DND status with value true is already set」 CiscoTermDNDStatusChangedEv の受信なし。	N.A.

DND–R

シナリオ 1

アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加します。管理ページまたは共通プロファイル ページから、端末 B で DND-R が有効にされます。

操作	イベント	コール情報
アプリケーションは端末 A および B に端末オブザーバを追加する。DND-R（サイレント）は、phone または admin ページにより端末 B 上で有効化される。 A は機能プライオリティ = 1（通常）で B に Call.connect を発行する。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL ConnFailedEv B Cause:. Cause: CAUSE_USERBUSY.	N.A.

シナリオ 2

アプリケーションは、Terminal.addObserver() を使用して、端末 A に端末オブザーバを追加します。管理ページまたは共通プロファイル ページから、端末 B で DND-R が有効にされています。

操作	イベント	コール情報
アプリケーションは端末 A および B に端末オブザーバを追加する。DND-R（サイレント）は、phone または admin ページにより端末 B 上で有効化される。 A は機能プライオリティ = 3（緊急）で B に Call.connect を発行する。	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A Called: B

シナリオ 3

DND（サイレント）: CFB を設定しないコール拒否。

操作	イベント	コール情報
アプリケーションは端末 A および B に端末オブザーバを追加する。DND-R は、CFB を設定せずに phone または admin ページにより端末 B 上で有効化される。 端末 A は端末 B に Call.connect を発行する。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL ConnFailedEv B Cause: CAUSE_USERBUSY.	NA

シナリオ 4

DND：CFB を C に設定したコール拒否。

操作	イベント	コール情報
アプリケーションは端末 A、B、C を監視している。 DND-R は、CFB を端末 C に設定して、端末 B 上で有効化される。 端末 A は端末 B に Call.connect を発行する。 コールが端末 B に表示されず、端末 C に転送される。	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for C cause: REDIRECTED CALL ConnInProgressEv for C Cause: REDIRECTED CALL CallCtlConnOfferedEv for C Cause: REDIRECTED CALL ConnAlertingEv for C Cause REDIRECTED CALL CallCtlConnAlertingEv for C Cause: REDIRECTED CALL TermConnCreatedEv for C Cause: REDIRECTED CALL TermConnRingingEv for C Cause: REDIRECTED CALL CallCtlTermConnTalkingEv Cause: CAUSE_REDIRECTED	Calling: A Called: C LastRedirectedParty: B

セキュア会議

操作	イベント	コール情報
シナリオ 1 設定：A（セキュア）および B（セキュア）。 A が B にコールする。B が応答する。 アプリケーションは、次を発行する。 getCallSecurtyStatus().	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1 このコールのコール セキュリティ ステータスを返す。	Calling: A Called: B CallSecurityStatus = 3 (ENCRYPTED) コール情報で更新される。 CallSecurityStatus = 3 (ENCRYPTED).
設定：A（セキュア）、B（セキュア）、および C（非セキュア）。 アプリケーションが、enableSecurtyStatusChangedEv () を発行して、ini parameter = true を設定する。 A が B にコールする。B が応答する。 B が C にコールする。C が応答する。 B が会議を開催する。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1.	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED). 注意：CallSecurityStatus=NotAuthenticated であっても、A と B は相互間にセキュアなメディアおよび会議ブリッジを持ち続ける。つまり、両者は暗号化された通話者であるため SRTP キー情報を受信し続ける。
シナリオ 3 設定：A（セキュア）、B（セキュア）、および C（セキュア）。 アプリケーションが、enableSecurtyStatusChangedEv () を発行して、ini parameter = true を設定する。 A が B にコールする。B が応答する。 B が C にコールする。C が応答する。 B が会議を開催する。 アプリケーションは、getCallSecurtyStatus() を発行する。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1. このコールのコール セキュリティ ステータス（Secure）を返す。	Participants: A, B, C CallSecurityStatus = 3 (ENCRYPTED) コール情報で更新される。

操作	イベント	コール情報
シナリオ 4 アプリケーションは、A、B、C にコール オブザーバを追加しない。 設定：A（セキュア）、B（セキュア）、および C（非セキュア）。 A が B にコールする。B が応答する。 B が C にコールする。C が応答する。 B が会議を開催する。 アプリケーションは、後で、A、B、C にコール オブザーバを追加する。 アプリケーションは、getCallSecurtyStatus() を発行する。	CallSecurityStatusChangedEv for callID=GC1 with Cause=CAUSE_SNAPSHOT このコールのコール セキュリティ ステータスを返す。	Participants: A, B, C CallSecurityStatus = 1 (NOTAUTHENTICATED) コール情報で更新される。
シナリオ 5 設定：A（セキュア）および B（セキュア）。 アプリケーションが、enableSecurtyStatusChangedEv() を発行して、 ini parameter = true を設定する。 A が B にコールする。B が応答する。 B がコールを保留にする。 B が、コールを再開する。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1 CallSecurityStatusChangedEv for callID=GC1 CallSecurityStatusChangedEv for callID=GC1	CallSecurityStatus = 3 (ENCRYPTED) コール情報で更新される。 CallSecurityStatus = 0 (UNKNOWN) コール情報で更新される。 CallSecurityStatus = (ENCRYPTED) コール情報で更新される。

操作	イベント	コール情報
シナリオ 6 設定：A（セキュア）、B（セキュア）、および C（非セキュア）。 アプリケーションが、enableSecurtyStatusChangedEv() を発行して、 ini parameter = true を設定する。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL	CallSecurityStatus = 3 (ENCRYPTED) GC1 のコール情報で更新される。
A が B にコールする。B が応答する。	CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1	CallSecurityStatus = 1 (NOTAUTHENTICATED) GC2 のコール情報で更新される。 CallSecurityStatus = 1 (NOTAUTHENTICATED) GC1 のコール情報で更新される。
B が C にコンサルト コールする。C が応答する。	CallActiveEv for callID=GC2 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL	
B は転送を実行する。	CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC2 CallCtlSecurityStatusChangedEv for callID=GC1	

操作	イベント	コール情報
シナリオ 7 設定：A（セキュア）、B（セキュア）、C（セキュア）、D（セキュア）、および E（認証済み）。 アプリケーションが、enableSecurtyStatusChangedEv() を発行して、ini parameter = true を設定する。 A、B、および C は、電話会議 1 に参加。 C、D、および E は別の電話会議 2 に参加。 C は、2 つの会議をチェーニングする。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for C' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC2:ConnCreatedEv for C' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for D' Cause: CAUSE_NORMAL GC2:ConnCreatedEv for E' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallCtlSecurityStatusChangedEv for callID=GC1	CallSecurityStatus = 3 (ENCRYPTED) GC1 のコール情報で更新される。 CallSecurityStatus = 2 (AUTHENTICATED) GC2 のコール情報で更新される。 CallSecurityStatus = 1 (NOTAUTHENTICATED) GC1 および GC2 のコール情報で更新される。
シナリオ 8 設定：A（セキュア）、B（セキュア）、B（認証済み） アプリケーションが、enableSecurtyStatusChangedEv() を発行して、ini parameter = true を設定する。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL GC1:ConnCreatedEv for A' Cause: CAUSE_NORMAL GC1:ConnCreatedEv for B' Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv CallSecurityStatusChangedEv for callID=GC1.	CallSecurityStatus = 2 (AUTHENTICATED) GC1 のコール情報で更新される。 注意：B' にコール オブザーバを追加したアプリケーションもまたイベントを受信する。つまり、新規イベントは、RIU 通話者にも配信される。

JTAPI Cisco Unified IP 7931G Phone の対話

A および C は、JTAPI アプリケーションで制御可能なアドレスです。B1 および B2 は、Cisco Unified IP 7931G Terminal 上のアドレスです。Cisco Unified IP 7931G Terminal は、アドレス間転送を実行するように設定されています。B1 と B2 は、それぞれ共用回線 B1' と B2' を持ち、JTAPI で制御可能な端末上に設定されています。

操作	イベント	コール情報
シナリオ 1 アプリケーションは、A を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは Cisco Unified IP 7931G Phone の transfer キーを押し、C にダイヤルする。B2 から C にコールが開始される。B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	A の CallObserver が受信する JTAPI イベント GC-1 CiscoTransferStartedEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, TransferredCall= null) GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling=A Called=B1 CurrCalling=A CurrCalled=B1 LRP=B1

操作	イベント	コール情報
シナリオ 2 アプリケーションは、A および B1' を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは Cisco Unified IP 7931G Phone の transfer キーを押し、C にダイヤルする。B2 から C にコールが開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	A および B1' の CallObserver が受信する JTAPI イベント GC-1 CiscoTransferStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, TransferredCall= null) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 CiscoTransferEndEv	Calling=A Called=B1 CurrCalling=A CurrCalled=B1 LRP=B1

操作	イベント	コール情報
シナリオ 3 アプリケーションは、A、B1'、および B2' を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは Cisco Unified IP 7931G Phone の transfer キーを押し、C にダイヤルする。B2 から C にコールが開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	A および B1' の CallObserver が受信する JTAPI イベント GC-1 CiscoTransferStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, TransferredCall= GC2) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	Calling=A Called=B1 CurrCalling=A CurrCalled=B1 LRP=B1

操作	イベント	コール情報
	GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv	

操作	イベント	コール情報
シナリオ 4 アプリケーションは、A、B1'、B2'、および C を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは Cisco Unified IP 7931G Phone の transfer キーを押し、C にダイヤルする。B2 から C にコールが開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	A、B1'、B2'、および C の CallObserver が受信する JTAPI イベント GC-1 CiscoTransferStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, TransferredCall= GC2) GC1- TermConnDroppedEv for TB1' Cause: CAUSE_NORMAL GC1- CallCtlTermConnDroppedEv for TB1' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC-1 ConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	Calling=A Called=B1 CurrCalling=A CurrCalled=B1 LRP=B1

操作	イベント	コール情報
	GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoTransferEndEv	

操作	イベント	コール情報
シナリオ 5 アプリケーションは、C を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の transfer キーを押し、 C をダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	C の CallObserver が受信する JTAPI イベント GC1- CallActiveEv for callID=101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoTransferStartEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, TransferredCall= GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER	Calling=B2 Called=C CurrCalling=A CurrCalled=C LRP=B1

操作	イベント	コール情報
	GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL	

操作	イベント	コール情報
	GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER CallControlCause: CAUSE_TRANSFER	

操作	イベント	コール情報
シナリオ 6 アプリケーションは、A と C の両方を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の transfer キーを押し、 C をダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、transfer キーを押し、転送を実行する。	A および C のオブザーバにおける JTAPI イベント GC-1CiscoTransferStartEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, TransferredCall= GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- 1 CiscoTransferEndEv Cause: CAUSE_NORMAL	Calling=A Called=B1 CurrCalling=A CurrCalled=C LRP=B1

操作	イベント	コール情報
	GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRANSFER NEW META GC-1 ConnDisconnectedEv for B1 –GC1 Cause: CAUSE_UNKNOWN GC-1 CallCtlConnDisconnectedEv for B1 – GC1 Cause: CAUSE_UNKNOWN CallControlCause: CAUSE_TRANSFER	
シナリオ 7 アプリケーションは、A を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、 C をダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	A の CallObserver が受信する JTAPI イベント GC-1 CiscoConferenceStartedEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, ConsultCall= null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC-1 CiscoConferenceEndEv	Calling=A Called=B1 CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
シナリオ 8 アプリケーションは、A および B1' を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、 C をダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	A の CallObserver が受信する JTAPI イベント GC-1 CiscoConferenceStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, ConsultCall= null) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv TB1' GC1 CallCtlTermConnBridgedEv TB1' GC-1 CiscoConferenceEndEv	Calling=A Called=B1 CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
シナリオ 9 アプリケーションは、A、B1'、および B2' を を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、C にダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	A、B1’、および B2’ CallObserver が受信する JTAPI イベント GC-1 CiscoConferenceStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, ConsultCall= GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv – TB1' GC1 CallCtlTermConnBridgedEv – TB1' GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE	Calling=A Called=B1 CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
	GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoConferenceEndEv	

操作	イベント	コール情報
シナリオ 10 アプリケーションは、A、B1'、B2'、および C を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、C にダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	A、B1'、B2'、および C の CallObserver が受信する JTAPI イベント GC-1 CiscoConferenceStartedEv (ControllerAddress=B1, ControllerTerminalConnection= TC at TB1', FinalCall=GC1, ConsultCall= GC2) GC-1 ConnCreatedEv for C Cause: CAUSE_NORMAL GC-1 ConnConnectedEv for C Cause: CAUSE_NORMAL GC-1 CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1 TermConnPassiveEv - TB1' GC1 CallCtlTermConnBridgedEv - TB1' GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- TermConnDroppedEv for TB2' Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TB2' Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE	Calling=A Called=B1 CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
	GC2- TermConnDroppedEv for TC Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for TC Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC-1 CiscoConferenceEndEv	

操作	イベント	コール情報
シナリオ 11 アプリケーションは、C を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、C にダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	C の CallObserver が受信する JTAPI イベント GC1- CallActiveEv for callID=101 Cause: CAUSE_NEW_CALL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnCreatedEv for B2 Cause: CAUSE_NORMAL GC-1CiscoConferenceStartEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, ConsultCall= GC2) Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnConnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE	Calling=B2 Called=C CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
	GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC1- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for A Cause: CAUSE_NORMAL GC1- ConnConnectedEv for A Cause: CAUSE_NORMAL	

操作	イベント	コール情報
	GC1- CallCtlConnEstablishedEv for A Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- ConnCreatedEv for B1 Cause: CAUSE_NORMAL GC1- ConnConnectedEv for B1 Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for B1 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	

操作	イベント	コール情報
シナリオ 12 アプリケーションは、A と C の両方を監視している。 A が B1 にコールし、B1 が応答する：GC1 ユーザは、Cisco Unified IP 7931G Phone の conference キーを押し、C にダイヤルする。コールは B2 から C に開始される。 B2 が C にコールし、C が応答する：GC2 ユーザは、conference キーを押し、会議を開催する。	A および C のオブザーバにおける JTAPI イベント GC-1CiscoConferenceStartEv (ControllerAddress=B1, ControllerTerminalConnection= Null, FinalCall=GC1, ConsultCall= GC2) Cause: CAUSE_NORMAL GC2- CiscoCallChangedEv for C Cause: CAUSE_NORMAL GC2- ConnDisconnectedEv for B2 Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for B2 Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- TermConnDroppedEv for CT Cause: CAUSE_NORMAL GC2- CallCtlTermConnDroppedEv for CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_TRAN CAUSE_CONFERENCE SFER GC2- ConnDisconnectedEv for C Cause: CAUSE_NORMAL GC2- CallCtlConnDisconnectedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC2- CallInvalidEv Cause: CAUSE_NORMAL GC1- ConnCreatedEv for C Cause: CAUSE_NORMAL GC1- ConnConnectedEv for C Cause: CAUSE_NORMAL GC1- CallCtlConnEstablishedEv for C Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE	Calling=A Called=B1 CurrCalling=A CurrCalled= Conference LRP=B1

操作	イベント	コール情報
	GC1- TermConnCreatedEv CT Cause: Other: 31 GC1- TermConnActiveEv CT Cause: CAUSE_NORMAL GC1- CallCtlTermConnTalkingEv CT Cause: CAUSE_NORMAL CallControlCause: CAUSE_CONFERENCE GC1- 1 CiscoConferenceEndEv Cause: CAUSE_NORMAL	

ロケール インフラストラクチャ変更シナリオ

シナリオ 1：JTAPI クライアント マシンに CallManager TFTP サーバへの接続がある

- インストール時、JTAPI クライアントはユーザに TFTP IP アドレスの入力を求めることがある。
- TFTP-IP アドレスは JTAPI.ini パラメータに保存される。
- JTAPI Preferences アプリケーションが初めて実行されると、ユーザは言語を選択するための言語タブに誘導される。
- ユーザは JTAPI Preferences アプリケーションを実行する言語を選択できる。
- JTAPI Preferences アプリケーションが 2 回目に実行されると、前にユーザが選択した言語で UI が表示される。

シナリオ 2：JTAPI クライアント マシンに CallManager TFTP サーバへの接続がない

- インストール時、JTAPI クライアントはユーザに TFTP-IP アドレスの入力を求めることがある。
- TFTP-IP アドレスは JTAPI.ini パラメータに保存される。
- JTAPI Preferences アプリケーションが初めて実行されると、ユーザは言語を選択するための言語タブに誘導されるが、ユーザが選択できるのは英語だけである。
- JTAPI Preferences アプリケーションが 2 回目に実行されると、英語で UI が表示される。
- TFTP 接続が復元される。ここで JTAPI Preferences UI が実行されると、ユーザが言語の選択に誘導される。

シナリオ 3：JTAPI クライアント マシンには CallManager TFTP サーバへの接続がある

- インストール時、JTAPI クライアントはユーザに TFTP-IP アドレスの入力を求めることがある。
- TFTP-IP アドレスは JTAPI.ini パラメータに保存される。
- JTAPI Preferences アプリケーションが初めて実行されると、ユーザは言語を選択するための言語タブに誘導される。
- ユーザは JTAPI Preferences アプリケーションを実行する言語を選択できる。
- JTAPI Preferences アプリケーションが 2 回目に実行されると、前にユーザが選択した言語で UI が表示される。
- ここで、新しい言語のサポートが追加された新しいロケール ファイルを使用できる。

- ユーザが JTAPI Preferences アプリケーションを実行すると、JTAPI 初期設定アプリケーションは、ユーザに使用可能であることを通知する。
- アプリケーションは JTAPI Preferences アプリケーションを再起動し、ユーザは新しい言語でサポートされる。

発信側の正規化

シナリオ 1：PSTN 番号（ローカル）から JTAPI によって監視される端末への着信コール

操作	イベント	コール情報
PSTN 番号 [55555555] A からコールが提供され、番号タイプはゲートウェイから JTAPI によって監視される端末 [2222] B への [Subscriber] である。	NEW META EVENT_____ META_CALL_STARTINGCallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (55555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfoNumber Type().getNumberType() は Subscriber を返す

シナリオ 2：国内 PSTN 番号（ローカル）から JTAPI によって監視される端末への着信コール

操作	イベント	コール情報
Dallas PSTN 番号 [55555555] A からコールが発信され、番号タイプはゲートウェイから JTAPI によって監視される端末 [2222] B への [National] である。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (97255555555) Called: B (2222) getModifiedCallingAddress (): 97255555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +197255555555 getCurrentCallingPartyInfoNumberType().getNumberType() は National を返す

シナリオ 3：国際 PSTN 番号から JTAPI によって監視される端末への着信コール

操作	イベント	コール情報
India PSTN 番号 [918028520261] からコールが発信され、番号タイプは San Jose ゲートウェイから JTAPI によって監視される端末 [2222] への [Inter-national] である。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (918028520261) Called: B (2222) getModifiedCallingAddress (): 918028520261 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +918028520261 getCurrentCallingPartyInfoNumberType().getNumberType() は Inter-National を返す

シナリオ 4：JTAPI によって監視される端末から PSTN 番号 [SUBSCRIBER] への発信コール

操作	イベント	コール情報
コールは JTAPI によって監視される端末 2222 から San Jose ゲートウェイを経由して PSTN 番号 [44444444] に発信され、番号タイプは [SUBSCRIBER] である。	NEW META EVENT_____META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (44444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 44444444 getCurrentCalledAddress(): 44444444 getCurrentCalledPartyInfo(): 44444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfoNumberType ().getNumberType () は Unknown を返す

シナリオ 5：JTAPI によって監視される端末から 国内 PSTN 番号への発信コール

操作	イベント	コール情報
コールは JTAPI によって監視される端末 2222 から San Jose ゲートウェイを経由して Dallas PSTN 番号 [97244444444] に発信され、番号タイプは [NATIONAL] である。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (97244444444) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 97244444444 getCurrentCalledAddress(): 97244444444 getCurrentCalledPartyInfo(): 97244444444 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfoNumber Type().getNumberType() は Unknown を返す

シナリオ 6：JTAPI によって監視される端末から国際 PSTN 番号への発信コール

操作	イベント	コール情報
コールは JTAPI によって監視される端末 2222 から San Jose ゲートウェイを経由して India PSTN 番号 [918028520261] に発信され、番号タイプは [INTERNATIONAL] である。	NEW META EVENT_____ META_CALL_STARTING CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL	Calling: A (2222) Called: B (918028520261) getModifiedCallingAddress (): 2222 getModifiedCalledAddress (): 918028520261 getCurrentCalledAddress():918028520261 getCurrentCalledPartyInfo(): 918028520261 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfoNumberType().getNumberType() は Unknown を返す

シナリオ 7 ：JTAPI によって監視される端末によって、他の PSTN にリダイレクトされる PSTN からの着信コール

操作	イベント	コール情報
コールが PSTN [55555555] から San Jose 経由で発信される。	NEW META EVENT_____ META_CALL_STARTING	Calling: A (55555555) Called: B (2222)

操作	イベント	コール情報
他の San Jose PSTN [44444444] にコールをリダイレクトする、JTAPI によって監視される端末 [2222]。 CallState [Idle] では、fwdDestinationAddress（リダイレクト アドレス）はマイナス（-）になる。	CallActiveEv for callID=GC1 Cause: CAUSE_NEW_CALL ConnCreatedEv for A Cause: CAUSE_NORMAL ConnConnectedEv for A Cause: CAUSE_NORMAL CallCtlConnInitiatedEv for A Cause: CAUSE_NORMAL TermConnCreatedEv for A Cause: CAUSE_NORMAL TernConnActiveEv for A Cause: CAUSE_NORMAL CallCtlConnDialingEv for A Cause: CAUSE_NORMAL CallCtlConnEstabilishedEv for A Cause: CAUSE_NORMAL ConnCreatedEv for B cause: CAUSE_NORMAL ConnInProgressEv for B Cause: CAUSE_NORMAL CallCtlConnOfferedEv for B Cause: CAUSE_NORMAL ConnAlertingEv for B Cause: CAUSE_NORMAL CallCtlConnAlertingEv for B Cause: CAUSE_NORMAL	getModifiedCallingAddress (): 55555555 getModifiedCalledAddress (): 2222 getCurrentCalledAddress(): 2222 getCurrentCalledPartyInfo(): 2222 getGlobalizedCallingParty (): +140855555555 getCurrentCallingPartyInfoNumberType().getNumberType() は SUBSCRIBER を返す destinationAddress: 44444444. getCurrentCallingPartyInfoNumberType().getNumberType() は Unknown を返す
	TermConnCreatedEv for B Cause: CAUSE_NORMAL TermConnRingingEv for B Cause: CAUSE_NORMAL CallCtlTermConnTalkingEv Cause: CAUSE_NORMAL CallRedirectReq Redirect Address = C CallRedirectRes ConnCreatedEv at C Cause: CAUSE_REDIRECTED ConnInProgress Calling party: A, Called Party: C, LRP: B CallRedirectRes CallStateChangedEv (IDLE) Reason: REDIRECT	

シナリオ 8：PSTN 番号（ローカル）から、JTAPI によって監視される端末（他の JTAPI によって監視される端末に転送する）への着信コール

操作	イベント	コール情報
PSTN 番号 [55555555] A からコールが発信され、番号タイプは San Jose ゲートウェイを経由して、コールを別の JTAPI によって監視される [2222] B に転送する JTAPI によって監視される [1111] X への [Subscriber] である。	転送後： GC1: CiscoTransferStartEv ConnCreatedEv for B ConnConnectedEv for B CallCtlConnEstablishedEv for B TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv GC2: CiscoTransferStartEv TermConnDroppedEv for X ConnDisconnectedEv for X CallCtlConnDisconnectedEv for X CiscoTransferStartEv	転送後： Calling: A (55555555) Called: B (2222) getModifiedCallingAddress (): A (+140855555555) getModifiedCalledAddress (): B (2222.) getCurrentCalledAddress(): B (2222) getCurrentCalledPartyInfo(): B (2222) getGlobalizedCallingParty: A +140855555555 getCurrentCallingPartyInfoNumberType().getNumberType() は Subscriber を返す

クリック ツー会議

A、B、C、D はアドレスで、TermA、TermB、TermC、TermD は対応する端末です。

操作	イベント	コール情報
A と B はクリック ツー コールを使用して作成されたコール GC1 で通話中である。ユーザは C を電話会議に追加する。GC2 は C での初期コールである。アプリケーションは、C だけを監視している。	GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv C CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: ConnInProgressEv C CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL	callingAddress=unknown calledAddress=C CurrentCalling=unknown CurrentCalled=C
	GC2: CallCtlConnOfferedEv C CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL	

操作	イベント	コール情報
	GC2: ConnAlertingEv C CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlConnAlertingEv C CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnCreatedEv TermC GC2: TermConnRingingEv TermC CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnRingingEv TermC CiscoFeatureReason=REASON_CONFERENCE Cause: CAUSE_NORMAL CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv C GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermC GC1: TermConnRingingEv TermC CiscoCallChangedEv GC2->GC1 CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: TermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallCtlTermConnDroppedEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	
	GC2: ConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	

操作	イベント	コール情報
	GC2: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC2: CallInvalidEv CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnCreatedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnConnectedEv C CiscoFeatureReason =REASON_NORMALCause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv C CiscoFeatureReason =REASON_NORMALCause: CAUSE_NORMAL GC1: TermConnTalkingEv TermC CiscoFeatureReason =REASON_NORMALCause: CAUSE_NORMAL	
A はクリック ツー コールを使用して B をコールする：GC1。A はクリック ツー 会議を使用して C をコールに追加する。アプリケーションには、A にコール オブザーバがある。	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER GC1: ConnCreatedEv A CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL	Calling address: A Called address: B Current calling: A Current called: B Last redirecting party=null C の会議参加後、CallInfo は適用されない。

操作	イベント	コール情報
	GC1: ConnInProgressEv A CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL GC1: CallCtlConnOfferedEv A CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv A CiscoFeatureReason=REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv A CiscoFeatureReason=REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnCreatedEv TermA GC1: TermConnRingingEv TermA CiscoFeatureReason=REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlTermConnRingingEv TermA CiscoFeatureReason=REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv A CiscoFeatureReason =REASON_NORMAL cause: CAUSE_NORMAL GC1: CallCtlConnEstablishedEv A CiscoFeatureReason =REASON_NORMAL Cause: CAUSE_NORMAL TermConnActiveEv TermA CiscoFeatureReason =REASON_NORMAL Cause: CAUSE_NORMAL GC1: TermConnTalkingEv TermA CiscoFeatureReason =REASON_NORMAL Cause: CAUSE_NORMAL GC1: ConnCreatedEv B CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL GC1: ConnInProgressEv B CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL	
	GC1: CallCtlConnOfferedEv B CiscoFeatureReason= REASON_REFER Cause: CAUSE_NORMAL GC1: ConnAlertingEv B CiscoFeatureReason= REASON_NORMAL Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv B CiscoFeatureReason= REASON_NORMAL Cause: CAUSE_NORMAL GC1: GC1: ConnConnectedEv B CiscoFeatureReason = REASON_NORMAL: CAUSE_NORMAL	

操作	イベント	コール情報
	GC1: CallCtlConnEstablishedEv B CiscoFeatureReason = REASON_NORMAL Cause: CAUSE_NORMAL GC1: ConnCreatedEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnAlertingEv C CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnAlertingEv TermC CiscoFeatureReason = REASON_CLICK_TO_CONFERENCE Cause: CAUSE_NORMAL	
A が D-GC3 にコンサルト コールする。A が会議を開催する。アプリケーションが受け取るイベントはコンサルト会議のイベントと同じままである。	GC1: TermConnHeldEv TermA GC3: ConsultCallActiveEv GC3: ConnCreatedEv A GC3: ConnCreatedEv D GC3: CallCtlConnAlerting D GC3: ConnConnectedEv D GC3: CallCtlConnEstablishedEv B CiscoConferenceStartEv GC3->GC1 GC3: CallCtlConnDisconnectedEv A GC3: CallCtlConnDisconnectedEv D GC1: ConnCreatedEv D GC1: CallCtlConnEstablishedEv D GC1: TermConnTalkingEv TermA GC3: CallInvalidEv	コンサルト コール GC3：Calling address: A Called address: D 会議の開催後、CallInfo は適用されない。
ユーザはクリック ツー会議機能を使用して D をドロップする。 ユーザはクリック ツー会議インターフェイスを使用して C をドロップする。	CiscoConferenceEndEvent GC1: ConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: CallCtlConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	Calling address: A Called address: B

操作	イベント	コール情報
	GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	
会議のすべての相手をドロップする。	GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_REFER	Calling address: A Called address: B
A はクリック ツー コールを使用して B をコールする。ユーザはクリック ツー会議を使用して C を会議に追加する。	GC1: ConnCreatedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	GC2: Calling address=unknown Called address: C
クリック ツー会議を使用してすべての相手をドロップする。		
アプリケーションには、A、B、および C にコール オブザーバがある。	GC1: ConnInProgressEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	
	GC1: CallCtlConnOfferedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	
	GC1: ConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	
	GC1: CallCtlConnAlertingEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER	
	GC1: TermConnCreatedEv TermA	
	GC1: TermConnRingingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC1: CallCtlTermConnRingingEv TermA	
	GC1: ConnConnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC1: TermConnActiveEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC1: CallCtlTermConnTalkingEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC1: ConnCreatedEv B Cause: CAUSE_NORMAL CiscoFeatureReason:REASON_REFER	

操作	イベント	コール情報
	GC1: ConnInProgressEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: CallCtlConnOfferedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_REFER GC1: ConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnAlertingEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnCreatedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnRingingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: TermConnActiveEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	
	GC2: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnCreatedEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

操作	イベント	コール情報
	GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC2: CiscoCallChangedEv GC2->GC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	
	GC1: TermConnCreatedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason REASON_CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnRingingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE	

操作	イベント	コール情報
	GC2: ConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC2: CallInvalidEv Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: TermConnActiveEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: CallCtlTermConnTalkingEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE クリック ツー会議を使用してすべての相手をドロップする。 GC1: TermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermA Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE	
	GC1: ConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv A Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: TermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermC Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: ConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE	

操作	イベント	コール情報
	GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: TermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermB Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: ConnDisconnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason= REASON_CONFERENCE GC1: CallInvalidEv	
A はクリック ツー コールを使用して B をコールする：GC1。A はクリック ツー 会議を使用して C をコールに追加する。ユーザは相手 C をドロップする。アプリケーションには、C だけにコール オブザーバがある。	GC1: TermConnDroppedEv TermC CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlTermConnDroppedEv TermC CiscoFeatureReason= REASON_CONFERENCE GC1: ConnDisconnectedEv C CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv C CiscoFeatureReason= REASON_CONFERENCE GC1: ConnDisconnectedEv A CiscoFeatureReason= REASON_CONFERENCE	NA
	GC1: CallCtlConnDisconnectedEv A CiscoFeatureReason= REASON_CONFERENCE GC1: ConnDisconnectedEv B CiscoFeatureReason= REASON_CONFERENCE GC1: CallCtlConnDisconnectedEv B CiscoFeatureReason= REASON_CONFERENCE GC1: CallInvalidEv	
A は GC1 を使用して B をコールする。TermC1 と TermC2 にアドレス C が設定されている。アプリケーションには、C にコール オブザーバがある。 ユーザは C にクリック ツー会議を使用する。	GC2: CallActiveEv CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_ CONFERENCE GC2: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CONFERENCE	Calling=unknown Called=C Last redirecting=null

操作	イベント	コール情報
	GC2: ConnInProgressEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CONFERENCE GC2: CallCtlConnOfferedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CONFERENCE GC2: ConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC2: CallCtlConnAlertingEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC2: TermConnCreatedEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC2: TermConnRingingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC2: TermConnCreatedEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC2: TermConnRingingEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ NORMAL GC1: CallActiveEv Cause: CAUSE_NEW_CALL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: ConnCreatedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE	
	CiscoCallChangedEv GC2->GC1 TermConn TermC1 CiscoFeatureReason: REASON_CLICK_TO_CONFERENCE GC1: ConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: CallCtlConnAlertingEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE	

操作	イベント	コール情報
	CiscoCallChangedEv GC2->GC1 TermConn TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: TermConnCreatedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: TermConnRingingEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC2: CallCtlConnDisconnectedEv C Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE	
	GC2: TermConnDroppedEv TermC1 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC2: TermConnDroppedEv TermC2 Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC2: CallInvalidEv GC1: ConnCreatedEv B Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: ConnConnectedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv B Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE	
	GC1: ConnCreatedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: ConnConnectedEv A Cause:CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: CallCtlConnEstablishedEv A Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_ CLICK_TO_CONFERENCE GC1: ConnConnectedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlConnEstablishedEv C Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnTalkingEv TermC1 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	C は TermC1 で応答する。

操作	イベント	コール情報
	GC1: TermConnPassEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL GC1: CallCtlTermConnInUseEv TermC2 Cause: CAUSE_NORMAL CiscoFeatureReason: REASON_NORMAL	

コール ピックアップ

基本のシナリオは次のようになります。

- B と C はコール ピックアップ グループのデバイスである。A はそのグループに含まれないデバイスである。
- A が B にコールする。
- C はオフフックになり、[Pickup] ソフトキー（[ピック] ソフトキー）を押す。
- C は A と通話中になる。

各デバイスは次のように監視しました。

- A、B、および C を監視
- A と B を監視
- A と C を監視
- B と C を監視
- A だけを監視
- B だけを監視
- C だけを監視

C だけを監視することは、お客様がシナリオで行っていたことであったため、この修正では特に配慮しました。この機能要求の目的は、C だけを監視している場合に、ピックアップ時に元の着信側（A）に関する情報を取得できるようにすることでした。すべてのテスト ケースが合格し、それらのすべてについて、正しい情報が表示されました。

これらのテスト ケースは、自動ピックアップを有効にした場合と無効にした場合で実行され、この 2 つの機能は大きく違いました。ほとんどのテスト ケースを下に示します。

A から B への「基本コール」は、すべてのケースで同じであるため、下の最初のケースでだけ示しています。

シナリオ 1

すべてのデバイスを監視し、自動ピックアップを有効にします。

操作	イベント	コール情報（GCID 情報）
A はオフフックになり、B をダイヤルする（基本コール）。B が呼び出し中になる。C はオフフックになり、[Pickup] ソフトキーを押す。C の古い接続はドロップされ、クリーンアップされる。Call 1 で C の接続が確立される。	GC1-CallActiveEvent-NONE	Calling: A, CCalled: NONE Calling: A, Called: NONE
	GC1-ConnCreatedEvent-A	CAUSE_NEW_CALL
	GC1-ConnConnectedEvent-A	REASON_NORMAL
	GC1-CallCtlConnInitiatedEv-A	LRP: NONE
	GC1-TermConnCreatedEvent	CCalling: A, CCalled: B
	GC1-TermConnActiveEvent	Calling: A, Called: B
	GC1-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NONE
	GC1-CallCtlConnDialingEv-A	CAUSE_NEW_CALL
	GC1-CallCtlConnEstablishedEv-A	REASON_NORMAL
	GC1-ConnCreatedEvent-B	LRP: NONE
	GC1-ConnInprogressEvent-B	REASON_CALLPICKUP
	GC1-CallCtlConnOfferedEv-B	CCalling: A, CCalled: C
	GC1-ConnAlertingEvent-B	LRP: NONE
	GC1-CallCtlConnAlertingEv	REASON_CALLPICKUP
	GC1-TermConnCreatedEvent	CCalling: C, CCalled: NONE
	GC1-TermConnRingingEvent	LRP: NONE
	GC1-CallCtlTermConnRingingEv	REASON_NORMAL
	GC2-CallActiveEvent-NONE	REASON_CALLPICKUP
	GC2-ConnCreatedEvent-C	CCalling: A, CCalled: C
	GC2-ConnConnectedEvent-C	REASON_NORMAL
	GC2-CallCtlConnInitiatedEv-C	
	GC2-TermConnCreatedEvent	
	GC2-TermConnActiveEvent	
	GC2-CallCtlTermConnTalkingEv	
	GC2-CiscoCallChangedEv	
	GC1-ConnCreatedEvent-C	
	GC1-ConnConnectedEvent-C	
	GC1-CallCtlConnInitiatedEv-C	
	GC1-TermConnCreatedEvent	
	GC1-TermConnActiveEvent	
	GC1-CallCtlTermConnTalkingEv	
	GC2-TermConnDroppedEv	
	GC2-CallCtlTermConnDroppedEv	
	GC2-ConnDisconnectedEvent-C	
	GC2-CallCtlConnDisconnectedEv-C	

操作	イベント	コール情報（GCID 情報）
	GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallCtlConnEstablishedEv-C	

（注）　次のシナリオの B と C は共にまったく同じ動作とイベントを発生します。デバイス C（コールをピックアップする）の動作だけ異なります。

シナリオ 2

自動ピックアップを無効にしてすべてのデバイスを監視します。

操作	イベント	コール ID 情報
C はオフフックになり、[Pickup] ソフトキーを押す。 コール 2 がドロップされるか無効になる。 C は Call 1 で接続を取得する。 B は Call 1 からドロップする。 C が呼び出し中になる。 C は A と通話している。	GC2-CallActiveEvent GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling C, CCalled: NONE LRP: NONE REASON_NORMAL REASON_CALLPICKUP CCalling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP Calling A, CCalled: C Calling: A, Called: C, LRP: B REASON_CALLPICKUP REASON_NORMAL REASON_NORMAL

自動ピックアップ オプションを有効にしている場合と無効にしている場合で、イベントのフローは大きく異なります。自動コール ピックアップが無効にされており、ユーザが [Pickup] ソフトキー（C）を押すと、電話が鳴ります。ユーザは通常のコールの場合と同様に電話に応答する必要があります。電話が鳴っていて、それらがオフフックになったときに、作成された元のコールが破棄されると、既存のコールに接続され、古い相手（B）がコールから削除されます。「Auto Call Pickup」が無効にされている場合、コールが変更されず、C が新しいコールに参加する前に破棄されるため、CiscoCallChangedEv は生成されません。

グループ ピックアップ シナリオは次のようになります。この場合、[Pickup] ソフトキーの代わりに [Group Pickup] ソフトキー（[G ピック] ソフトキー）が使われます。これは、実際にピックアップ グループの番号をダイヤルする必要があります。さらに、グループ ピックアップには、Auto Call Pickup

サービス パラメータが必要です。一般的なフローとコール イベントは、通常のコール ピックアップのシナリオと同じですが、ピックアップ番号の必要なダイヤルに関するイベントが追加されます。グループ ピックアップによる変更を明確に示すために、これらの追加のイベントを表に太字で示しています。

シナリオ 3

グループ ピックアップと自動ピックアップを有効にしてすべてのデバイスを監視します。

操作	コール イベント	コール ID 情報
C はオフフックになり、[Group Pickup] ソフトキーを押す。	GC1（明確にするために他に追加） GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NONE LRP: NONE REASON_NORMAL
C は PU 番号をダイヤルする。		CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU
C が元のコールに追加される。 ピックアップが元のコールに追加される。	**GC2-CallCtlConnDialingEv-C** **GC2-ConnCreatedEvent-PU** **GC2-ConnInprogressEvent-PU** **GC2-CallCtlConnEstablishedEv-C** GC2-CiscoCallChangedEv	CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP
ピックアップ番号が Call 2 から削除される。 C が Call 2 からドロップされる。	GC1-ConnCreatedEvent-C **GC1-ConnCreatedEvent-PU** GC1-ConnConnectedEvent-C **GC1-CallCtlConnEstablishedEv-C** GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv **GC1-ConnInprogressEvent-PU** **GC1-CallCtlConnOfferedEv-PU**	CCalling: A, CCalled: C, LRP:B REASON_CALLPICKUP, LRP: PU CCalling: C, CCalled: PU REASON_CALLPICKUP
ピックアップ番号が Call 1 から削除される。 B がドロップされるか、無効にされる。	**GC2-ConnDisconnectedEvent-PU** **GC2-CallCtlConnDisconnectedEv-PU** GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP

上のグループ ピックアップのケースでは、変更はほんの少しであり、それらはすべてピックアップ番号のダイヤルのために特別に必要な手順に直接関連しています。

シナリオ 4

グループ ピックアップと自動ピックアップを無効にしてすべてのデバイスを監視します。

操作	イベント	コール情報
C はオフフックになり、[Group Pickup] ソフトキーを押した。	GC1	
	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL
C は PU 番号をダイヤルしている。	GC2-CallCtlTermConnTalkingEv	
	GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL
PU が Call 2 から削除される。	GC2-CallCtlConnEstablishedEv-C	CCalling: C, CCalled: PU
	GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent	CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP
C が Call 2 から削除される。 Call 2 が破棄される。 C は Call 1 で接続を取得する。	GC2-CallObservationEndedEv	
	GC1-ConnCreatedEvent[ADDRS] GC1-ConnInprogressEvent GC1-CallCtlConnOfferedEv	CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP
B は Call 1 からドロップする。	GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent GC1-CallCtlConnDisconnectedEv	CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP
C が呼び出し中になる。	GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent	REASON_NORMAL
C がピックアップする。	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_NORMAL

上の表は、すべてのデバイスが監視されたときのシナリオを示しています。さまざまなピックアップとグループ ピックアップで、可能なあらゆる組み合わせでデバイスが実行されました。一部のシナリオでは出力がまったく同じであり、重複しているものもあったため、それらはここに示していません。たとえば、デバイス A と B は同じであったため、1 回だけ示しています。

シナリオ 5

デバイス B だけを監視します。

操作	コール イベント	コール ID/ コール情報
A は B をコール中である。	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-TermConnCreatedEvent	CCalling: A, CCalled: B, Calling: A, Called: B, LRP: NONE REASON_NORMAL
B が呼び出し中になる。	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	
A が Call 1 から削除される。	GC1-ConnDisconnectedEvent-A GC1-CallCtlConnDisconnectedEv-A	REASON_CALLPICKUP
B が Call 1 から削除される。	GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-CallInvalidEvent GC1-CallObservationEndedEv	 REASON_NORMAL

シナリオ 6

デバイス A だけを監視します。

操作	コール イベント	コール ID/ コール情報
A はオフフックになり、B をダイヤルする。	GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnInitiatedEv-A GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-CallCtlConnDialingEv-A	CCalling: A, CCalled: NO, NO LRP REASON_NORMAL
B が呼び出し中になる。	GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEvent-B GC1-CallCtlConnOfferedEv-B GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B	CCalling A, CCalled B, Called: NOT SET LRP: NONE
C が呼び出し中になる。	GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C	CCalling: A, CCalled: C, LRP: B Called: NOT SET REASON_CALLPICKUP REASON_NORMAL
B が Call 1 から削除される。	GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C	REASON_CALLPICKUP REASON_NORMAL

シナリオ 7

自動ピックアップを有効にして、デバイス C だけを監視します。

操作	コール イベント	コール ID/ コール情報
C はオフフックになり、[Pickup] ホットキーを押す。	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv GC2-CiscoCallChangedEv GC1-CallActiveEvent-NONE GC1-ConnCreatedEvent-C GC1-ConnConnectedEvent-C GC1-CallCtlConnInitiatedEv GC1-TermConnCreatedEvent	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL REAON_CALLPICKUP CCalling A, CCalled: NONE LRP: NONE
C は Call 1 に接続される。	GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP
C は Call 2 からドロップする。	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C	CCalling: C, CCalled: NONE REASON_CALLPICKUP
Call 2 は無効化またはクリアされる。	GC2-CallInvalidEvent GC2-CallObservationEndedEv	REASON_CALLPICKUP
A と C は Call 1 で接続される。	GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-CallCtlConnEstablishedEv-C	CCalling A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL

シナリオ 8

自動ピックアップを無効にして、デバイス C だけを監視します。

操作	コール イベント	コール ID/ コール情報
C はオフフックになり、[Pickup] ソフトキーを押した。	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL
Call 2 が破棄される。	GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	REASON_CALLPICKUP CCalling: C, CCalled: NONE REASON_NORMAL
C が Call 1 に追加されるが、ピックアップしない。	GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent	CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL
C が呼び出し中になる。	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	
C がピックアップし、Call 1 に接続される。	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_NORMAL

シナリオ 9

グループ ピックアップと自動ピックアップを有効にしてデバイス C だけを監視します。

操作	コール イベント	コール ID/ コール情報
C はオフフックになり、[Pickup] ソフトキーを押す。	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL CCalling: C, CCalled: PU
C はピックアップ番号をダイヤルする。	GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C GC2-CiscoCallChangedEv GC1-CallActiveEvent	 CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL REASON_CALLPICKUP CCalling: A,C Called: C
C が Call 1 に追加される。 PU が Call 1 に追加される。	GC1-ConnCreatedEvent-C GC1-ConnCreatedEvent-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnCreatedEvent GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv GC1-ConnInprogressEvent-PU GC1-CallCtlConnOfferedEv-PU	 CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP
PU 番号が Call 2 から削除される。	GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv	 CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP
C は Call 2 から削除される。 Call 2 は無効化またはクリアされる。	GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	 CCalling C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL
C は Call 1 に接続される。	GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-PU GC1-CallCtlConnEstablishedEv-PU GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C	 CCalling: A, CCalled: C REASON_CALLPICKUP
PU が Call 1 から削除される。	GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU	 CCalling: A, CCalled: C REASON_CALLPICKUP

シナリオ 10

グループ ピックアップと自動ピックアップを無効にして、デバイス C だけを監視します。

操作	コール イベント	コール ID/ コール情報
C はオフフックになり、[Group Pickup] ソフトキーを押す。	GC2-CallActiveEvent-NONE GC2-ConnCreatedEvent-C GC2-ConnConnectedEvent-C GC2-CallCtlConnInitiatedEv-C GC2-TermConnCreatedEvent GC2-TermConnActiveEvent GC2-CallCtlTermConnTalkingEv	CCalling: C, CCalled: NO, NO LRP REASON_NORMAL
C は PU 番号をダイヤルする。	GC2-CallCtlConnDialingEv-C GC2-ConnCreatedEvent-PU GC2-ConnInprogressEvent-PU GC2-CallCtlConnEstablishedEv-C	CCalling: C, CCalled: PU, LRP: PU REASON_CALLPICKUP REASON_NORMAL
PU が Call 2 からドロップする。	GC2-ConnDisconnectedEvent-PU GC2-CallCtlConnDisconnectedEv-PU GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv	REASON_CALLPICKUP
C が Call 2 からドロップする。	GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C	
Call 2 が破棄される。	GC2-CallInvalidEvent	REASON_CALLPICKUP REASON_NOTMAL
C は Call 1 に追加される。	GC1-CallObservationEndedEv GC1-CallActiveEvent GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C GC1-ConnCreatedEvent-A GC1-ConnConnectedEvent-A GC1-CallCtlConnEstablishedEv-A GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C GC1-TermConnCreatedEvent	CCalling: A, CCalled: C, LRP: B REASON_CALLPICKUP REASON_NORMAL
C が呼び出し中になる。	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv	
C が A に接続される。	GC1-ConnConnectedEvent-C GC1-CallCtlConnEstablishedEv-C GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	

コーリングサーチスペースおよび機能プライオリティを使用した selectRoute()

コーリングサーチスペースおよび機能プライオリティを int. の配列として使用した selectRoute() API を次の表に示します。

操作	イベント	コール情報
電話 A、B、C、D にコール オブザーバを追加する。 ルート ポイント RP を登録する。 3 行の SelectRoute API を使用して、ルート コール バックを登録する。 選択したルート：A、.....CSS：0、FP： 1 選択したルート：B、.....CSS：1、FP： 3 選択したルート：D、.....CSS：1、FP： 1 C が RP をコールする。 A で呼び出し音が鳴る。 A が応答する。C-A コールが接続される。	GC1 CallActiveEv GC1 ConnCreatedEv C: GC1 ConnConnectedEv C GC1 CallCtlConnInitiatedEv C: GC1 TermConnCreatedEv TC GC1 TermConnActiveEv TC GC1 CallCtlTermConnTalkingEv TC GC1 CallCtlConnDialingEv C: GC1 CallCtlConnEstablishedEv C: GC1 ConnCreatedEv RP: GC1 ConnInProgressEv RP: GC1 CallCtlConnOfferedEv RP: リダイレクト要求の処理後 GC1 ConnCreatedEv A: GC1 ConnInProgressEv A: GC1 CallCtlConnOfferedEv A: GC1 ConnDisconnectedEv RP: GC1 CallCtlConnDisconnectedEv RP: GC1 ConnAlertingEv A: GC1 CallCtlConnAlertingEv A: GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnConnectedEv A: GC1 CallCtlConnEstablishedEv A: GC1 TermConnActiveEv A GC1 TermConnActiveEv A [C] CiscoRTPInputStartedEv [A] CiscoRTPOutputStartedEv [A] CiscoRTPInputStartedEv [C] CiscoRTPOutputStartedEv	calling: C lastRedirected:RP called: A

エクステンション モビリティ ログイン ユーザ名

端末 A はユーザのコントロール リストに含まれ、端末 B はユーザのコントロール リストに含まれていません。エクステンション モビリティ ログイン ユーザ名は John で、アプリケーションのエンド ユーザ ID は John です。

操作	結果	コール情報
プロバイダーを開く。端末 A にはオブザーバがない。アプリケーションは端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	InvalidStateException がスローされる。	NA
プロバイダーを開く。端末 A にオブザーバを追加する。アプリケーションは端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションはユーザ名として空の文字列 "" を受け取る。	NA
プロバイダーを開く。ユーザ「John」は端末 A に EMLogin し、オブザーバを端末 A に追加する。アプリケーションが端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA
ユーザ「John」は端末 A に EMLogin する。ここでプロバイダーを開く、オブザーバを端末 A に追加する。アプリケーションが端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA
ユーザ「John」は端末 A に EMLogin する。ここでプロバイダーを開く。オブザーバを端末 A に追加する。ユーザ「John」が端末 A から EMLogout する。アプリケーションが端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションはユーザ名として空の文字列 "" を受け取る。	NA
プロバイダーを開く。ユーザ「John」が端末 B に EMLogin する。オブザーバを追加する。アプリケーションが端末 B で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA
ユーザ「John」が端末 B に EMLogin する。OpenProvider を実行する。オブザーバを端末 B に追加する。アプリケーションが端末 B で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA

端末 A はユーザのコントロール リストに含まれ、ユーザ Kerry のエクステンション モビリティ ログアウト プロファイルが設定されています。Kerry プロファイルにはログアウト ユーザ名が Kerry として設定されています。ログイン ユーザ名 John の別のプロファイルがあります。

操作	結果	コール情報
ユーザ John は端末 A にログインし、Open Provider を実行して、オブザーバを端末 A に追加する。アプリケーションが端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA
John が端末 A でログアウトする。アプリケーションが端末 A で CiscoTerminal.getEMLoginUserName() を呼び出す。	アプリケーションは文字列「John」を取得する。	NA

発信側の IP アドレス

次にコール シナリオの例を示します。

基本的なコール シナリオ

- JTAPI アプリケーションは相手 B を監視する。
- 相手 A は IP フォンである。
- A が B にコールする。
- A の IP アドレスは相手 B を監視している JTAPI アプリケーションから使用できる。

コンサルト転送のシナリオ

- JTAPI アプリケーションは相手 C を監視する。
- 相手 B は IP フォンである。
- A は B と通話中である。
- B は C へのコンサルト転送コールを開始する。
- B の IP アドレスは相手 C を監視している JTAPI アプリケーションから使用できる。

コンサルト会議のシナリオ

- JTAPI アプリケーションは相手 C を監視する。
- 相手 B は IP フォンである。
- A は B と通話中である。
- B は C へのコンサルト会議コールを開始する。
- B の IP アドレスは相手 C を監視している JTAPI アプリケーションから使用できる。

リダイレクトのシナリオ

- JTAPI アプリケーションは相手 B と相手 C を監視する。
- 相手 A は IP フォンである。
- A が B にコールする。
- A の IP アドレスは相手 B を監視している JTAPI アプリケーションから使用できる。
- 相手 A が B を相手 C にリダイレクトする。

- 発信側の IP アドレスは、相手 B を監視している JTAPI アプリケーションから使用できない（サポートされないシナリオ）。
- B の IP アドレスのコールが相手 C を監視している JTAPI アプリケーションに提供される。

CiscoJtapiProperties

1) ソケット接続タイムアウトを 5 秒に設定し、プライマリ CTI マネージャのイーサネット ケーブルを引き抜き、通常のプロバイダー オープンを実行します。予想される結果：プライマリ CTI マネージャへのソケット接続が 5 秒以内に失敗します。

2) ソケット接続タイムアウトを 5 秒に設定し、プライマリ CTI マネージャのイーサネット ケーブルを引き抜き、セキュリティ オプションを True に設定して、セキュリティ保護されたプロバイダー オープンを実行します。予想される結果：プライマリ CTI マネージャへのソケット接続が 5 秒以内に失敗します（ソケット接続が 5 秒以内にタイム アウトします。ただし、最初はセキュリティ証明書の確認のため、いくらか余分に時間がかかります）。

3) ソケット接続タイムアウトを 0 秒に設定し、プライマリ CTI マネージャのイーサネット ケーブルを引き抜き、セキュリティ オプションを True に設定して、セキュリティ保護されたプロバイダー オープンを実行します。予想される結果：プライマリ CTI マネージャへのソケット接続で新しいサービス パラメータが使われなくなります（ソケット接続が 23 秒以内にタイム アウトします。ただし、最初はセキュリティ証明書の確認のため、いくらか余分に時間がかかります）。

IPv6 のサポート

使用例 1：基本的なコール シナリオ：発信側が IPv6 対応電話機、着信側が IPv6

操作	イベント	コール情報 / 予想される結果
IPv6 対応の電話機 A が、JTAPI によって監視される IPv6 対応デバイス B を、GC1 を使用して呼び出す。	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、A の IPv6 形式アドレスを InetAddress オブジェクトとして返す。 getCallingPartyIpAddr() は、null を返す。 CiscoOutputStartedEv の CiscoRTPOutputProperties の getRemoteAddress() は、（A の）遠端 Ipv6 RTP アドレスを含む。 CiscoRTPInputStartedEv の CiscoRTPInputProperties の getRemoteAddress() は、監視対象電話（B）の Ipv6 RTP アドレスを含む。

B が応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 2：基本的なコール シナリオ：発信側が IPv6 対応電話機、着信側が IPv4

操作	イベント	コール情報/予想される結果
IPv6 対応の電話機 A が、JTAPI によって監視される IPv4 対応デバイス B を、GC1 を使用してコールする。	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、A の IPv6 形式アドレスを InetAddress オブジェクトとして返す。 getCallingPartyIpAddr() は、null を返す。 CiscoRTPOutputStartedEv の CiscoRTPOutputProperties の getRemoteAddress() は、Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv4 RTP アドレスを含む。 CiscoRTPInputStartedEv の CiscoRTPInputProperties の getLocalAddress() は、監視対象電話の Ipv4 RTP アドレスを含む。

B が応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 3：基本的なコール シナリオ：発信側が IPv4 対応電話機、着信側が IPv6

操作	イベント	コール情報 / 予想される結果
IPv4 対応の電話機 A が、JTAPI によって監視される IPv6 対応デバイス B を、GC1 を使用してコールする。	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は、InetAddress オブジェクト内 A の IPv4 形式アドレスを返す。 CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、null を返す。 CiscoRTPOutputStartedEv の CiscoRTPOutputProperties の getRemoteAddress() は、Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv6 RTP アドレスを含む。 CiscoRTPInputStartedEv の CiscoRTPInputProperties の getLocalAddress() は、監視対象電話の Ipv6 RTP アドレスを含む。

B が応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 4：基本的なコール シナリオ：発信側が IPv4 対応電話機、着信側が IPv4

操作	イベント	コール情報 / 予想される結果
IPv4 対応の電話機 A が、JTAPI によって監視される IPv4 対応デバイス B を、GC1 を使用してコールする。	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() は、InetAddress オブジェクト内 A の IPv4 形式アドレスを返す。 CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr_v6() は、null を返す。 CiscoRTPOutputStartedEv の CiscoRTPOutputProperties の getRemoteAddress() は、遠端 Ipv4 RTP アドレスを含む。 CiscoRTPInputStartedEv の CiscoRTPInputProperties の getLocalAddress() は、監視対象電話の Ipv4 RTP アドレスを含む。

B が応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 5：コンサルト転送のシナリオ：IPv6 デバイス コンサルト

操作	イベント	コール情報 / 予想される結果
GC1：A と B の間のコール コンサルト コール： IPv6 対応の電話機 B から、JTAPI によって監視されるデバイス C に、GC2 を使用しての転送を打診する。	NEW META EVENT_____META_CALL_STARTING	コンサルト コールでは次のようになります。 CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、InetAddress オブジェクト内の B の IPv6 形式アドレスを、C を監視している JTAPI アプリケーションに返す。 その一方で、CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は、null を返す。
C が応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 6：コンサルト転送のシナリオ：IPv4 デバイス コンサルト

操作	イベント	コール情報 / 予想される結果
GC1：A と B の間のコール コンサルト コール： IPv4 対応の電話機 B から、JTAPI によって監視されるデバイス C に、GC2 を使用しての転送を打診する。	NEW META EVENT_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は、InetAddress オブジェクト内の B の IPv4 形式アドレスを、C を監視している JTAPI アプリケーションに返す。 その一方で、CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、null を返す。
C が応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 7：リダイレクトのシナリオ

操作	イベント	コール情報/予想される結果
GC1：A(IPv6) と B の間のコール リダイレクト コール： 電話機 B から、JTAPI によって監視されるデバイス C に、GC2 を使用してコールをリダイレクトする。	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、InetAddress オブジェクト内の A の IPv6 形式アドレスを、C を監視している JTAPI アプリケーションに返す。 その一方で、CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は、null を返す。
C が応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL	

操作	イベント	コール情報 / 予想される結果
	TermConnRingingEv for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 8：リダイレクトのシナリオ（IPv4）

操作	イベント	コール情報 / 予想される結果
GC1：A(IPv4) と B の間のコール リダイレクト コール： 電話機 B から、JTAPI によって監視されるデバイス C に、GC2 を使用してコールをリダイレクトする。	New Meta Event_____META_CALL_STARTING	CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は、InetAddress オブジェクト内の A の IPv4 形式アドレスを、C を監視している JTAPI アプリケーションに返す。 その一方で、CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、null を返す。
C が応答する。	**CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for B Cause: CAUSE_NORMAL **ConnConnectedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause: CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TernConnActiveEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for B Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for B Cause : CAUSE_NORMAL	

操作	イベント	コール情報 / 予想される結果
	ConnCreatedEv for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for B Cause : CAUSE_NORMAL **ConnDisconnectedEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for B Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 9：リダイレクトのシナリオ：発信デバイスのリダイレクト

操作	イベント	コール情報 / 予想される結果
GC1: A が B（IPv4）をコールする。 リダイレクト コール： 電話機 A から、JTAPI によって監視されるデバイス C に、GC2 を使用してコールをリダイレクトする。	New Meta Event_____META_CALL_STARTING **CallActiveEv** for callID=GC2 Cause: CAUSE_NEW_CALL	CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() は、InetAddress オブジェクト内の B の IPv4 形式アドレスを、C を監視している JTAPI アプリケーションに返す。 CiscoCallCtlConnOfferedEv. getCallingPartyIpAddr() または CiscoCallCtlConnOfferedEv. リダイレクトのあと、getCallingPartyIpAddr()_v6 は、InetAddress 内の A の IP アドレスを、B を監視している JTAPI アプリケーションに**返さない。**

C が応答する。	**ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B Cause : CAUSE_NORMAL **CallCtlConnEstablishedEv** for B Cause : CAUSE_NORMAL **ConnCreatedEv** for C cause : CAUSE_NORMAL **ConnInProgressEv** for C Cause : CAUSE_NORMAL **CallCtlConnOfferedEv** for C Cause : CAUSE_NORMAL **ConnAlertingEv** for C Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for C Cause : CAUSE_NORMAL **TermConnCreatedEv** for C Cause : CAUSE_NORMAL **TermConnRingingEv** for C Cause : CAUSE_NORMAL **TermConnDroppedEv** for A Cause : CAUSE_NORMAL **ConnDisconnectedEv** for A Cause : CAUSE_NORMAL **CallCtlTermConnDroppedEv** for A Cause : CAUSE_REDIRECTED **ConnConnectedEv** for C Cause : CAUSE_NORMAL **TermConnActiveEv** for C Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 10：ルート シナリオ：IPv6 対応機が RoutePoint をコールし IPv6 デバイスにルート コールする

操作	イベント	コール情報 / 予想される結果
IPv6 対応の電話機 A が、GC1 を使用して、JTAPI によって監視される IPv6 対応デバイス B にそのコールを経路選択する RoutePoint をコールする。	NEW META EVENT_____META_CALL_STARTING	CiscoRouteEvent.getCallingPartyIpAddr_v6() は、A の IPv6 形式アドレスを InetAddress オブジェクトとして返す。 その一方で、CiscoRouteEvent.getCallingPartyIpAddr() は、null を返す。 CiscoRTPOutputStartedEv の CiscoRTPOutputProperties の getRemoteAddress() は、遠端 Ipv6 RTP アドレスを含む。 CiscoRTPInputStartedEv の CiscoRTPInputProperties の getLocalAddress() は、監視対象電話の Ipv6 RTP アドレスを含む。

B が応答する。	**CallActiveEv** for callID=GC1 Cause: CAUSE_NEW_CALL **ConnCreatedEv** for A Cause: CAUSE_NORMAL **ConnConnectedEv** for A Cause : CAUSE_NORMAL **CallCtlConnInitiatedEv** for A Cause: CAUSE_NORMAL **TermConnCreatedEv** for A Cause : CAUSE_NORMAL **TernConnActiveEv** for A Cause : CAUSE_NORMAL **CallCtlConnDialingEv** for A Cause : CAUSE_NORMAL **CallCtlConnEstabilishedEv** for A Cause : CAUSE_NORMAL **ConnCreatedEv** for B cause : CAUSE_NORMAL **ConnInProgressEv** for B Cause : CAUSE_NORMAL **CallRouteEv** for B Cause : CAUSE_NORMAL **ConnAlertingEv** for B Cause : CAUSE_NORMAL **CallCtlConnAlertingEv** for B Cause : CAUSE_NORMAL **TermConnCreatedEv** for B Cause : CAUSE_NORMAL **TermConnRingingEv** for B Cause : CAUSE_NORMAL **CallCtlTermConnTalkingEv** Cause : CAUSE_NORMAL	

使用例 11：エンタープライズ パラメータ「Enable IPv6」が有効

次の CTI Manager IP のリストを提供すると、アプリケーションによってプロバイダーがオープンされます。

- CTI Manager1 の IPv4 アドレス
- CTI Manager1 の IPv6 アドレス
- CTI Manager2 の IPv4 アドレス
- CTI Manager2 の IPv6 アドレス

これで、JTAPI は、いったん CTI Manager と接続を確立できますが、後に CTI Manager1 がダウンすると、フェールオーバーの試行で、アプリケーションから見て接続に遅延が発生します。JTAPI は最初に CTI Manger1 の IPv6 アドレス（リストの次）に接続しようとするため、その IP アドレスが同じ CTI Manager のもので、タイムアウトになったのが 1 度だけだとしても、CTI Manager2 の IPv4 アドレスを使用して試行し、成功します（CTI Manager2 が実行中と仮定）。

プロバイダー オープン シナリオ

1. 再接続試行のサービス パラメータは、セットされておらず（あるいは 0 にセットされていて）、エンタープライズ パラメータ「Enable IPv6」は無効化されています。アプリケーションは、IPv4 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI マネージャで接続をオープンできるようになります。
   - CTI Manager が停止 : JTAPI は、CTI Manager が再起動されて接続が復元されるまで、CTI マネージャに無期限に再接続しようとします。
   - エンタープライズ パラメータ「Enable IPv6」は有効化され、CTI マネージャが再起動されます。JTAPI は、同じ IPv4 アドレスで CTI Manager に再接続できるようになります。
2. 再接続試行のサービス パラメータは、セットされておらず（あるいは 0 にセットされていて）、エンタープライズ パラメータ「Enable IPv6」は有効化されています。アプリケーションは、IPv4 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI Manager で接続をオープンできるようになります。
   - CTI Manager が停止 : JTAPI は、CTI Manager が再起動されて接続が復元されるまで、CTI マネージャに無期限に再接続しようとします。
   - エンタープライズ パラメータ「Enable IPv6」は無効化され、CTI マネージャが再起動されます。JTAPI は、同じ IPv4 アドレスで CTI Manager に再接続できるようになります。ただし、IPv6 アドレスで登録された既存のデバイスは、新規の原因コード「IP_CAPABILITY_MISMATCH」で「CiscoTermRegistrationFailedEv」を使用してクローズされます。
3. 再接続試行のサービス パラメータは、セットされておらず（あるいは 0 にセットされていて）、エンタープライズ パラメータ「Enable IPv6」は有効化されています。アプリケーションは、IPv6 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI Manager で接続をオープンできるようになります。
   - CTI Manager が停止 : JTAPI は、CTI Manager が再起動されて接続が復元されるまで、CTI マネージャに無期限に再接続しようとします。
   - エンタープライズ パラメータ「Enable IPv6」は無効化され、CTI マネージャが再起動されます。JTAPI は、もう IPv6 アドレスをサポートしないため、CTI Manager に再接続できませんが、タイム サービス パラメータが再度有効になって CTI Service が再起動されるまで継続して再接続しようとします。
4. 再接続試行のサービス パラメータは、いずれかの整数（たとえば 5）にセットされ、エンタープライズ パラメータ「Enable IPv6」は無効化されています。アプリケーションは、IPv4 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI マネージャで接続をオープンできるようになります。
   - CTI Manager は停止 : JTAPI は、オープンしているデバイスおよびプロバイダーがすべてクローズするまで、CTI マネージャへの再接続を 5 回試行します。
   - エンタープライズ パラメータ「Enable IPv6」は有効化され、CTI マネージャが再起動されます。JTAPI は、同じ IPv4 アドレスで CTI Manager に再接続できるようになります。

5. 再接続試行のサービス パラメータは、いずれかの整数（たとえば 5）にセットされており、エンタープライズ パラメータ「Enable IPv6」は有効化されます。アプリケーションは、IPv4 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI Manager で接続をオープンできるようになります。
   - CTI Manager は停止：JTAPI は、オープンしているデバイスおよびプロバイダーがすべてクローズするまで、CTI マネージャへの再接続を 5 回試行します。
   - エンタープライズ パラメータ「Enable IPv6」は無効化され、CTI マネージャが再起動されます。JTAPI は、同じ IPv4 アドレスで CTI Manager に再接続できるようになります。ただし、IPv6 アドレスで登録された既存のデバイスは、新規の原因コード「IP_CAPABILITY_MISMATCH」で「CiscoTermRegistrationFailedEv」を使用してクローズされます。
6. 再接続試行のサービス パラメータは、いずれかの整数（たとえば 5）にセットされており、エンタープライズ パラメータ「Enable IPv6」は有効化されます。アプリケーションは、IPv6 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI Manager で接続をオープンできるようになります。
   - CTI Manager は停止：JTAPI は、オープンしているデバイスおよびプロバイダーがすべてクローズするまで、CTI マネージャへの再接続を 5 回試行します。
   - エンタープライズ パラメータ「Enable IPv6」は無効化され、CTI マネージャが再起動されます。JTAPI は、もう IPv6 アドレスをサポートしないため、CTI Manager に再接続できませんが、JTAPI は、すべてのデバイスおよびプロバイダーがクローズするまで、（同じことが再度 Cisco Unified CM で有効化できるため）CTI Manager にさらに 5 回再接続しようとします。
7. エンタープライズ パラメータ「Enable IPv6」が無効化されます。アプリケーションは、IPv6 アドレスでプロバイダーをオープンしようとします。JTAPI は、CTI マネージャで接続をオープンできません。再試行できるのは、接続がいったん確立できた場合だけですが、このシナリオでは最初の試行から失敗しているため、その後に再接続は試行されません。

   エンタープライズ パラメータ「Enable IPv6」が有効化されます。次の CTI Manager IP のリストを提供すると、アプリケーションによってプロバイダーがオープンされます。
   - CTI Manager1 の IPv4 アドレス
   - CTI Manager1 の IPv6 アドレス
   - CTI Manager2 の IPv4 アドレス
   - CTI Manager2 の IPv6 アドレス

   これで、JTAPI は、いったん CTI Manager と接続を確立できますが、後に CTI Manager1 がダウンすると、フェールオーバーの試行で、アプリケーションから見て接続に遅延が発生します。JTAPI は最初に CTI Manger1 の IPv6 アドレス（リストの次）に接続しようとするため、その IP アドレスが同じ CTI Manager のもので、タイムアウトになったのが 1 度だけだとしても、CTI Manager2 の IPv4 アドレスを使用して試行し、成功します（CTI Manager2 が実行中と仮定）。

発信側 IP アドレスのシナリオ

1. Ipv6 対応の電話機が、CTI 制御可能デバイスをコールします。続いて、CTI 制御可能デバイスが、JTAPI アプリケーションによって監視されます。JTAPI は、Ipv6 発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（ルートポイント以外）または CiscoRouteEvent（ルートポイント）の通知を生成します。
   getCallingPartyIpAddr() は、null を返します。
   getCallingPartyIpAddr_v6() は、実際の発信側 IPv6 アドレスを返します。

2. Ipv4 対応の電話機が、CTI 制御可能デバイスをコールします。続いて、CTI 制御可能デバイスが、JTAPI アプリケーションによって監視されます。JTAPI は、Ipv4 発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（ルートポイント以外）または CiscoRouteEvent（ルートポイント）の通知を生成します（既存の動作）。
getCallingPartyIpAddr() は、実際の発信側 IPv4 アドレスを返します。
getCallingPartyIpAddr_v6() は、null を返します。
3. Ipv6 電話は、すでに JTAPI アプリケーションによって監視されている CTI 制御可能デバイスだけをコールします。JTAPI は、Ipv6 発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（非ルートポイント）または CiscoRouteEvent（ルートポイント）の通知を生成します。
getCallingPartyIpAddr() は、null を返します。
getCallingPartyIpAddr_v6() は、実際の発信側 IPv6 アドレスを返します。
4. Ipv4 対応電話は、すでに JTAPI アプリケーションによって監視されている CTI 制御可能デバイスだけをコールします。JTAPI は、Ipv4 形式の発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（非ルートポイント）または CiscoRouteEvent（ルートポイント）の通知を生成します。
getCallingPartyIpAddr() は、実際の発信側 IPv4 アドレスを返します。
getCallingPartyIpAddr_v6() は、null を返します。
5. Ipv4_v6（デュアル スタック）電話は、CTI 制御可能デバイスをコールします。続いて、CTI 制御可能デバイスが、JTAPI アプリケーションによって監視されます。JTAPI は、Ipv4 および Ipv6 の発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（非ルートポイント）または CiscoRouteEvent（ルートポイント）の通知を生成します。
getCallingPartyIpAddr() は、実際の発信側 IPv4 アドレスを返します。
getCallingPartyIpAddr_v6() は、実際の発信側 IPv6 アドレスを返します。
6. Ipv4_v6（デュアル スタック）電話は、すでに JTAPI アプリケーションによって監視されている CTI 制御可能デバイスだけをコールします。JTAPI は、Ipv4 および Ipv6 の発信側 IP アドレスを含む CiscoCallCtlConnOfferedEv（非ルートポイント）または CiscoRouteEvent（ルートポイント）の通知を生成します。
getCallingPartyIpAddr() は、実際の発信側 IPv4 アドレスを返します。
getCallingPartyIpAddr_v6() は、実際の発信側 IPv6 アドレスを返します。

RTP アドレス

1. Ipv6 対応電話は、Ipv6 JTAPI によって監視される電話をコールし、その電話が応答します。JTAPI は、次のものを生成します。
   - 遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
2. Ipv4 対応電話は、Ipv4 JTAPI によって監視される電話をコールし、その電話が応答します。JTAPI は、次のものを生成します（既存の動作）。
   - 遠端 Ipv4 RTP アドレス を含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
3. Ipv4 対応電話は、Ipv6 JTAPI によって監視されるデバイスをコールし、その電話が応答します。JTAPI は、次のものを生成します。
   - Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
4. Ipv6 対応電話は、Ipv4 JTAPI によって監視されるデバイスをコールし、その電話が応答します。JTAPI は、次のものを生成します。

- Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
- 監視対象電話の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv

5. デュアル スタック（Ipv4_v6）電話は、他のデュアル スタック（Ipv4_v6) JTAPI によって監視されるデバイスをコールし、優先するメディアの停止が IPv6 にセットされ、そのコールに応答があると、JTAPI は次のものを生成します。
   - その発信側デバイスの遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
6. デュアル スタック（Ipv4_v6）電話が、他のデュアル スタック（Ipv4_v6) JTAPI によって監視されるデバイスをコールし、優先するメディアの停止が IPv4 にセットされ、そのコールに応答があってから、JTAPI は次のものを生成します。
   - その発信側デバイスの遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
7. デュアル スタック（Ipv4_v6）電話が、Ipv4 JTAPI によって監視されるデバイスをコールし、そのコールに応答があってから、JTAPI は次のものを生成します。
   - その発信側デバイスの遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
8. デュアル スタック（Ipv4_v6）電話が、Ipv6 JTAPI によって監視されるデバイスをコールし、そのコールに応答があってから、JTAPI は次のものを生成します。
   - その発信側デバイスの遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
9. IPv4 電話が、デュアル スタック（Ipv4_v6）JTAPI によって監視されるデバイスをコールし、そのコールに応答があってから、JTAPI は次のものを生成します。
   - その発信側デバイスの遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
   - 監視対象電話の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
10. IPv6 電話が、デュアル スタック（Ipv4_v6）JTAPI によって監視されるデバイスをコールし、そのコールに応答があってから、JTAPI は次のものを生成します。
    - その発信側デバイスの遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
    - 監視対象電話の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
11. JTAPI によって監視される IPv6 電話（A）が、JTAPI によって監視される IPv4 電話（B）をコールします。B が応答し、IPv6 電話（C）に転送を打診します。C が応答し、B が転送を実行すると、JTAPI は次のものを生成します。

- A：
  - C の遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
  - A の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
- C：
  - A の遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
  - C の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv

12. JTAPI によって監視される IPv4 電話（A）が、JTAPI によって監視される IPv4 電話（B）をコールします。B が応答し、IPv6 電話（C）に転送を打診します。C が応答し、B が転送を実行すると、JTAPI は次のものを生成します。

- A：
  - Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
  - A の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
- C：
  - Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
  - C の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv

13. JTAPI によって監視される IPv6 電話（A）が、JTAPI によって監視される IPv4 電話（B）をコールします。B が応答し、IPv6 電話（C）に会議を打診します。C が応答し、B が会議を開催します。会議ブリッジには、IPv4 アドレスがあります。その後、JTAPI は、次のものを生成します。

- A：
  - Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
  - A の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv
- B：
  - 会議ブリッジの遠端 Ipv4 RTP アドレスを含む CiscoRTPOutputStartedEv
  - B の Ipv4 RTP アドレスを含む CiscoRTPInputStartedEv
- C：
  - Call Manager に自動的に挿入され Ipv4/Ipv6 変換を実行する MTP に対応する遠端 Ipv6 RTP アドレスを含む CiscoRTPOutputStartedEv
  - C の Ipv6 RTP アドレスを含む CiscoRTPInputStartedEv

CTI ポート/ルート ポイント登録シナリオ

1. CTI ポート/ルート ポイントには、「IPv4_v6」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv6 アドレスと IPv6 としてのアプリケーション アドレッシング機能を使用して、その CTI ポート/ルート ポイントの CTIManager への静的登録を試行します。登録が成功し、CTI ポート/ルート ポイントは、IPv6 アドレスで CTIManager を使用して登録されます。
2. CTI ポート/ルート ポイントには、「IPv4_v6」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv4 アドレスと IPv4 としてのアプリケーション アドレッシング機能を使用して、その CTI ポート/ルート ポイントの CTIManager への静的登録を試行します。登録が成功し、CTI ポート/ルート ポイントは、IPv4 アドレスで CTIManager を使用して登録されます。
3. CTI ポート/ルート ポイントには、「IPv4_v6」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv4 および IPv6 のアドレスと IPv4_v6 としてのアプリケーション アドレッシング機能を使用して、その CTI ポート/ルート ポイントの CTIManager への静的登録を試行します。登録が成功し、CTI ポート/ルート ポイントは、IPv4 および IPv6 のアドレスで CTIManager を使用して登録されます。
4. CTI ポート/ルート ポイントには、「IPv4 のみ」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv4 アドレスと IPv4 としてのアプリケーション アドレッシング機能を使用して、その CTI ポート/ルート ポイントの CTIManager への静的登録を試行します。登録が成功し、CTI ポート/ルート ポイントは、IPv4 アドレスで CTIManager を使用して登録されます。

5. CTI ポート/ルート ポイントには、「IPv6 のみ」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv6 アドレスと IPv6 としてのアプリケーション アドレッシング機能を使用して、その CTI ポート/ルート ポイントの CTIManager への静的登録を試行します。登録が成功し、CTI ポート/ルート ポイントは、IPv6 アドレスで CTIManager を使用して登録されます。
6. CTI ポート/ルート ポイントには、「IPv4 のみ」として設定された「IP アドレッシング モード」があります。アプリケーションは、IPv6 アドレスの提供と、IPv6（または Ipv4_v6）のみとしてのアプリケーション アドレッシング機能の通知のいずれか、または両方によって、その CTI ポート/ルート ポイントの静的登録を試行し、その後、CiscoRegistrationExceptionas で失敗します。
7. CTI ポート/ルート ポイントには、「IPv6 のみ」として設定された「IP アドレッシング モード」があります。アプリケーションは、その CTI ポート/ルート ポイントを CTIManager に動的に登録しようとします。登録時にそのアプリケーションによって通知された IP 機能は、IPv4（または IPv4_v6）だけです。その後、その要求は CiscoRegistrationException で拒否されます。
8. CTI ポート/ルート ポイントには、「IPv4 のみ（または IPv4 と v6 両方」として設定された「IP アドレッシング モード」があります。アプリケーションは、その CTI ポート/ルート ポイントを CTIManager に動的に登録しようとします。登録時にそのアプリケーションによって通知された IP 機能は、IPv4 のみです。その後、登録は成功し、CTI ポート/ルート ポイントは、SetRTPParams 要求によって同じものが提供されたときに、IPv4 アドレスで登録されます。
9. CTI ポート/ルート ポイントには、「IPv6 のみ（または IPv4 と v6 両方」として設定された「IP アドレッシング モード」があります。アプリケーションは、その CTI ポート/ルート ポイントを CTIManager に動的に登録しようとします。登録時にそのアプリケーションによって通知された IP 機能は、IPv6 のみです。その後、登録は成功し、CTI ポート/ルート ポイントは、SetRTPParams 要求によって同じものが提供されたときに、IPv6 アドレスで登録されます。
10. CTI ポート/ルート ポイントには、「IPv4_v6 両方」として設定された「IP アドレッシング モード」があります。アプリケーションは、その CTI ポート/ルート ポイントを CTIManager に動的に登録しようとします。登録時にそのアプリケーションによって通知された IP 機能は、IPv4 と v6 両方です。その後、登録は成功し、CTI ポート/ルート ポイントは、SetRTPParams 要求によって同じものが提供されたときに、IPv4_v6 アドレスで登録されます。
11. アプリケーションが、その IP 機能を通知することで CTI ポート/ルート ポイントを IPv6 として動的に登録しようとしても、それはすでに IPv4 アドレスで他のアプリケーションに登録されています。それから、その要求は CiscoRegistrationException で拒否されるか、または「CiscoTermRegistrationFailedEv」が新しい原因コード「IP_CAPABILITY_MISMATCH」と共に送信されます。

拡張テスト ケース

1. アプリケーションは IPv4 アドレスで、エンタープライズ パラメータ「Enable IPv6」が有効化されている CTI Manager に、プロバイダーをオープンします。アプリケーションは、アプリケーション アドレッシング機能を「IPv6 のみ」として通知することで、デバイス IP アドレッシング モードが「IPv4_v6」にセットされている IPv6 アドレスで、CTI ポート/ルート ポイントの登録を試みます。登録要求は成功します。
2. JTAPI によって監視される IPv6 デバイス A が、他の JTAPI によって監視される IPv4 デバイス B をコールし、コールが提供されて B で応答されます。その場合、CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr() は null を返します。CiscoCallCtlConnOfferedEv.getCallingPartyIpAddr_v6() は、実際の発信側 IPv6 アドレスを返します。

- B：
  - CiscoRTPInputStartedEv は、B の IPv4 アドレスを持つようになります。

– CiscoRTPOutputStartedEv は、MTP リソースの IPv4 アドレスを持つようになります。

ここで興味深いのは、発信側 IP アドレスが IPv6 アドレスであるのに対し、CiscoRTPOutputStartedEv が IPv4 アドレスを持つことです。

回線をまたいで直接転送（Direct Transfer Across Lines）の使用例

操作	イベント	コール情報 / 予想される結果
アプリケーションは A、B1、B2、および C（B1 と B2 は同じ端末 TB 上の 2 つのアドレス）を監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。
アプリケーションは A、B1、B2、および C（B1 と B2 は、この機能をサポートする電話で回線をまたいだ手動の接続転送を可能にしている同じ端末上の 2 つのアドレス）を監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2		CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。

ユーザ B2 は、電話の UI から transfer キーを押してアクティブ コール（A→B コール）を選択し、回線をまたいだ接続転送のために、transfer キーを再度押す。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B2 GC3: ConnConnectedEv for B2 GC3: CallCtlConnInitiatedEv for B2 GC3: TermConnCreatedEv for TB2 GC3: TermConnActiveEvent for TB2 GC3: CallCtlTermConnTalkingEv for TB2	
ユーザは、回線をまたいだ接続転送を実行するために、transfer キーを再度押す。	GC3: TermConnDroppedEv for TB2 GC3: CallCtlTermConnDroppedEv for TB2 GC3: ConnDisconnectedEv for B2 GC3: CallCtlConnDisconnectedEv for B2 GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

アプリケーションが A、B1、B2 を監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。
アプリケーションが B1、B2 を監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	GC1: CiscoTransferStartEv GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEv GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEv GC1: CallObservationEndedEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。

アプリケーションは、B1 だけを監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	JTAPI は、PlatformException「Transfer controller is not set and could not find a suitable TerminalConnection」をスローする。JTAPI が GC2 から B2 のコール レッグを取得/検出できないため。	
アプリケーションは、A だけを監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 ユーザは、電話の UI から transfer キーを押してアクティブ コール（A→B コール）を選択し、回線をまたいだ接続転送のために、transfer キーを再度押す。	GC2: CallActiveEvent GC2: ConnCreatedEv for A GC2: ConnCreatedEv for C GC1: CiscoCallChangedEv GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for A GC2: TermConnActiveEvent for A GC2: CallCtlTermConnTalkingEv for A GC2: ConnConnectedEv for C GC2: CallCtlConnEstablishedEv for C GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC1: TermConnDroppedEv for A GC1: CallCtlTermConnDroppedEv for A GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: CallInvalidEvent GC1: CallObservationEndedEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。
アプリケーションは、B2 だけを監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	JTAPI は、GC1 から B1 のコール レッグを取得/検出できないため、PlatformException「Transfer controller is not set and could not find a suitable TerminalConnection」をスローする。	

アプリケーションは、C だけを監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 ユーザは、電話の UI から transfer キーを押してアクティブ コール（A→B コール）を選択し、回線をまたいだ接続転送のために、transfer キーを再度押す。	GC2: CiscoTransferStartEv GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: CiscoTransferEndEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。
新しいユーザ ロール（Standard Supports Connected Xfer/Conf）が、アプリケーション ユーザに関連付けられる。 アプリケーションがプロバイダーをオープンして、上記のユーザ ロールを関連付け解除する。	JTAPI は、次のものを配信する。 ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoTermRestrictedEv CiscoAddrRestrictedEv （回線をまたいで接続された tx/conf をサポートしているすべての電話が対象）	CiscoProviderCapabilityChangedEv.hasConnectedTransferConferenceCapabilityChanged() は、True を返す。

アプリケーションは A、B1、B2、C および C'（B1 と B2 は同じ端末 TB 上の 2 つのアドレス、C' は C の SharedLine）を監視している。 A が B1 にコールし、B1 が応答する：GC1 B2 が C にコールし、C が応答する：GC2 B1 に setTransferController GC1.transfer(GC2)	転送時： GC1: CiscoTransferStartEv GC2: CiscoCallChangedEv GC1: ConnCreatedEv for C GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC' GC1: TermConnPassiveEvent for TC' GC1: CallCtlTermConnInUseEv for TC' GC2: TermConnDroppedEv for TC' GC2: CallCtlTermConnDroppedEv for TC' GC2: CiscoCallChangedEv GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv for TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	CiscoTransferStartEv.getControllerTerminalName() は、B1 と B2 の端末名を返す。

新しいユーザ ロール（Standard Supports Connected Xfer/Conf）が、アプリケーション ユーザに関連付けられない。 アプリケーションは、回線をまたいで接続された転送 / 会議を許可する電話で、オブザーバの追加を試みる。	回線をまたいで接続された転送 / 会議を許可する電話は、制限付きとして公開される。 JTAPI は、PlatformExceptionImpl（「Terminal is restricted」、CiscoJtapiException.CTIERR_DEVICE_RESTRICTED）をスローする。	CiscoTerminal.isRestricted() が TRUE を返す。
新しいユーザ ロール（Standard Supports Connected Xfer/Conf）が、アプリケーション ユーザに関連付けられない。 アプリケーションがプロバイダーをオープンして、上記のユーザ ロールを関連付ける。	JTAPI は、次のものを配信する。 ProvInServiceEv CiscoProviderCapabilityChangedEv CiscoAddrActivatedEv CiscoTermActivatedEv （回線をまたいで接続された tx/conf をサポートしているすべての電話が対象）	CiscoProviderCapabilityChangedEv.hasConnectedTransferConferenceCapabilityChanged() は、True を返す。

Connected Conference または回線をまたいで参加（Join Across Lines）の使用例：新しい電話の動作

操作	イベント	コール情報 / 予想される結果
Connected Conference Across Lines をサポートする電話を制御する新規権限「Standard Supports Connected Xfer/Conf」は、ユーザと関連付けられていない。 電話 TA（回線 A）、TB（回線 B1、B2）、および T3（回線 C）。TC は、Connected Conference Across Lines を許可する電話。 すべてを監視。 GC1: A が B1 にコールする。 GC2: B2 が C にコールする。		

Connected Conference Across Lines を、（この機能をサポートしている）電話 TB で、GC1 および GC2 に対し、手動で行う。	アプリケーションは、TB、B1 および B2 にオブザーバを追加するときに、PlatformExceptionImpl（「Terminal is restricted」、CiscoJtapiException.CTIERR_DEVICE_RESTRICTED）を取得する。 GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	CiscoConferenceStartEv.getControllerAddress() は、B1 を返す。 CiscoConferenceStartEv.getControllerTerminalName() は、TB を返す。

拡張された MWI の使用例

操作	結果
アプリケーションは、CiscoAddress.setMessageSummary() を呼び出して、拡張されたメッセージ受信カウントをサポートする電話に、音声および FAX のカウントをセットする。	電話は、提供され更新された音声および FAX のカウントを表示し、また、それに伴って、MWI インジケータも更新する。正常な応答が返ってくる。
アプリケーションは、CiscoAddress.setMessageSummary() を呼び出して、拡張されたメッセージ受信カウントをサポートしない電話に、音声および FAX のカウントをセットする。	それに伴って、電話は MWI インジケータの更新だけを行う。電話には、音声カウントも FAX カウントも表示されない。正常な応答が返ってくる。

アプリケーションは、CiscoAddress.setMessageSummary() を呼び出して、音声カウントをセットするが、提供される「高優先度の」音声カウントは、提供される「トータルの」音声カウントより大きい。	要求は失敗し、次のエラーが返される。 INVALID_HIGH_PRIORITY_VOICE_COUNTS
アプリケーションは、CiscoAddress.setMessageSummary() を呼び出して、FAX カウントをセットするが、提供される「トータルの」FAX カウントが、許容最大サイズよりも大きい。	要求は失敗し、次のエラーが返される。 INVALID_TOTAL_FAX_COUNTS

回線をまたいで参加（Join Across Lines）の拡張機能

A、C、D、E、および F は異なる端末のアドレスです。B1 と B2 は同じ端末 TermB のアドレスです。

A、B1、および C は、B1 がコントローラである電話会議 GC1 に参加し、コンファレンス ブリッジ Conference-1 に接続されています。B2、D、および E は、D がコントローラである電話会議 GC2 に参加し、ブリッジ Conference-2 に接続されています。

操作	イベント
アプリケーションは GC1.conference(GC2) を呼び出して 2 つの電話会議をチェーンして、B1 および B2 で 2 つのコールの会議を開く。	**CallObserver の A、C、B1 へのイベント：** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv GC1 Ev.getAddedConnection は、Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-2 への接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC1 を返す。 **B2、D、E での CallObserver のイベント：** ConnDisconnectedEv B2 GC2 Cause=NORMAL CallCtlConnDisconnectedEv B2 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-1 と Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC1 と GC2 を返す。

アプリケーションは GC2.conference (GC1) を呼び出して、2 つの電話会議をチェーンする。	**B2、D、E での CallObserver のイベント：** TermConnActiveEv TermB GC2 CallCtlTermConnTalkingEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC2 を返す。 **A、B1、C での CallObservers のイベント：** ConnDisconnectedEv B1 GC1 Cause=NORMAL CallCtlConnDisconnectedEv B1 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC1 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は、Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC1 を返す。

A、B1、C は conference-1（GC1）に参加し、B1、D、E は conference-2（GC2）に参加し、B2、F、G は conference-3（GC-3）に参加している。 アプリケーションは GC1.conference(GC2, GC3) を開始し、B1 をコントローラとして設定して、会議を開催する。	**A、B1、C での CallObserver のイベント：** TermConnActiveEv TermB GC1 CallCtlTermConnTalkingEv TermB GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-2 GC1 ConnConnectedEv Conference-2 GC1 CallCtlConnEstablishedEv Conference-2 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は、Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC1 を返す。 TermConnDroppedEv TermB GC2 CallCtlTermConnDroppedEv TermB GC2 ConnCreatedEv Conference-3 GC1 ConnConnectedEv Conference-3 GC1 CallCtlConnEstablishedEv Conference-3 GC1 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC1 Ev.getAddedConnection は、Conference-3 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-2 と Conference-3 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC2 と GC3 を返す。 **Event for CallObserver at B1,D & E:** ConnDisconnectedEv B1 GC2 Cause=NORMAL CallCtlConnDisconnectedEv B1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC2 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE

	ConnCreatedEv Conference-1 GC2 ConnConnectedEv Conference-1 GC2 CallCtlConnEstablishedEv Conference-1 GC2 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC2 Ev.getAddedConnection は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-1-GC2 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC2 を返す。 **B2、F、G での CallObserver のイベント：** ConnDisconnectedEv B2 GC3 Cause=NORMAL CallCtlConnDisconnectedEv B2 GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE TermConnDroppedEv TermB GC3 Cause=NORMAL CallCtlTermConnDroppedEv TermB GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE ConnCreatedEv Conference-1 GC3 ConnConnectedEv Conference-1 GC3 CallCtlConnEstablishedEv Conference-1 GC3 Cause=NORMAL, callCtlCause=CAUSE_CONFERENCE CiscoConferenceChainAddedEv – GC3 Ev.getAddedConnection は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceConnections() は、Conference-1 の接続を返す。 Ev.getConferenceChain().getChainedConferenceCalls() は、GC3 を返す。

コール シナリオ：A、B1、C は、B1 がコントローラである電話会議に参加しています。B2 は、D と GC2 でコール中です。

アプリケーションはリクエスタを B2 と設定し、GC2.conference(GC1) をコールする。 getControllerAddress() は B2 を返す。 getOriginalControllerAddress() は B1 を返す。	A CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv TermB GC1 CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D ConnConnectedEv CiscoConferenceEndEv B2 ConnDisconnectedEv B GC2 CallCtlTermConnHeldEv TermB GC2 D CallActiveEv GC2 ConnAlertingEv D GC2 ConnConnectedEv D GC2 CiscoConferenceStartEv TermConnDroppedEv TermB GC2 CallActiveEv GC1 CiscoCallChangedEv TermConnTalkingEv TermB GC1 TermConnDroppedEv TermD GC2 CallObservationEndedEv GC2 CiscoConferenceEndEv
上の設定で、アプリケーションが要求コントローラとして B1 を使用する場合、 getControllerAddress() は B1 を返す。 getOriginalControllerAddress() は B1 を返す。	イベントは上記と同じ

スワップ/キャンセルおよび転送/会議の動作変更

使用例 1 接続された転送を許可する電話での接続された転送	GC1 と GC2 のコールが通常どおり作成される。	
A が B にコールし、B が応答する：GC1 B が A→B コールを保留にする。 B が C にコールし、C が応答する：GC2	GC1: CallCtlTermConnHeldEv for TB	
ユーザ B は、電話の UI から transfer キーを押してアクティブ コール（A→B コール）を選択する。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
ユーザ B は、transfer キーを再度押す。	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B GC2: CallCtlConnDisconnectedEv for B GC1: TermConnDroppedEv for TBGC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

使用例 2		
Connected Transfer on phone with SharedLine を持つ電話で接続された転送（A および A' は、SharedLine）	GC1 と GC2 のコールが通常どおり作成される。	
A が B にコールし、B が応答する：GC1	GC1: CallCtlTermConnHeldEv for TB	
B が A→B コールを保留にする。		
B が C にコールし、C が応答する：GC2	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
ユーザ B は、transfer キーを押してアクティブ コール（A→B コール）を選択する。	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoTransferStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA' GC2: TermConnPassiveEvent for TA' GC2: CallCtlTermConnInUseEv for TA' GC1: TermConnDroppedEv for TA' GC1: CallCtlTermConnDroppedEv for TA' GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for AGC2: TermConnDroppedEv for TB2 GC2: CallCtlTermConnDroppedEv for TB2 GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC1: TermConnDroppedEv for TB1 GC1: CallCtlTermConnDroppedEv for TB1 GC1: ConnDisconnectedEv for B1	
ユーザ B は、transfer キーを再度押す。		

	GC1: CallCtlConnDisconnectedEv for B1 GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	
使用例 3		
接続された転送/会議：キャンセル機能	GC1 と GC2 のコールが通常どおり作成される。 GC1: CallCtlTermConnHeldEv for TB	
A が B にコールし、B が応答する：GC1		
B が A→B コールを保留にする。		
B が C にコールし、C が応答する：GC2		
ユーザ B は、transfer ハードキーを押す。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
ユーザ B は、cancel キーを押す。	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getConsultCall() は null を返す。

使用例 4a		
接続された転送/会議：キャンセル機能	GC1 と GC2 のコールが通常どおり作成される。 GC1: CallCtlTermConnHeldEv for TB	
A が B にコールし、B が応答する：GC1		
B が A→B コールを保留にする。		
B が C にコールし、C が応答する：GC2		
ユーザ B は、transfer ハードキーを押す。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
ユーザは、select active calls キーを押す。	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv	
ユーザ B は、cancel キーを押す。	GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getConsultCall() は null を返す。

使用例 4b		
接続された転送/会議：キャンセル機能	GC1 と GC2 のコールが通常どおり作成される。 GC1: CallCtlTermConnHeldEv for TB	
A が B にコールし、B が応答する：GC1		
B が A→B コールを保留にする。		
B が C にコールし、C が応答する：GC2		
ユーザ B は、transfer（または conference）ハード キーを押す。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
ユーザは、active calls キーを選択し、GC1 も選択する（A→B コール）	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEv GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getConsultCall() は GC1 を返す。
ユーザ B は、cancel キーを押す。		

使用例 5		
コンサルト転送：スワップ コール	GC1 と GC2 のコールが通常どおり作成される。	getCiscoFeatureReason() は、CiscoFeatureReason.REASON_NORMAL を返す。
A が B にコールし、B が応答する：GC1	GC1: CallCtlTermConnHeldEv for TB	
B が A→B コールを保留にする。		
B は、コンサルト転送を C に設定し、C が応答する：GC2	GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB	
ユーザ B は、Swap キーを押す。	GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEv for C	
ユーザは B は、transfer キーを押し、転送を実行する。	GC1: ConnConnectedEv for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEv for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEv for C GC2: CallCtlConnDisconnectedEv for C GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEv for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC2: CallObservationEndedEv GC1: CiscoTransferEndEv	

使用例 6		
コンサルト転送：スワップ/キャンセル A が B にコールし、B が応答する：GC1 A が、A→B コールを保留にする。 B は、コンサルト転送を C に設定し、C が応答する：GC2 ユーザ B は、[Swap] ソフトキーを押す。 ユーザ B は、[Cancel] ソフトキーを押す。	GC1 と GC2 のコールが通常どおり作成される。 GC1: CallCtlTermConnHeldEv for TB For TA (GC2), CallCtlTermConnHeldEv For TA (GC1), CallCtlTermConnTalkingEv GC1: CiscoCallFeatureCancelledEv	getCiscoFeatureReason() は、CiscoFeatureReason.REASON_NORMAL を返す。 CiscoCallFeatureCancelledEv.getCall() は、GC1 を返す。 CiscoCallFeatureCancelledEv.getConsultCall() は、GC2 を返す。
使用例 7		
電話から開始されたコンサルト転送。アプリケーションは、転送/会議の設定要求を送信する。要求は失敗する。 A が B にコールし、B が応答する：GC1 B は転送コールを C に設定する。 B が C にコールし、C が応答する：GC2 アプリケーションは新しいコールを作成し、別の consult() 要求を送信する。	GC1 と GC2 のコールが通常どおり作成される。 要求は、PlatformException「CTIERR_CONSULTCALL_ALREADY_OUTSTANDING」で失敗する。	

使用例 8a		
コンサルト転送/会議：アプリケーションは、接続された転送/会議をサポートする電話での プライマリ コールを再開し、別のコンサルト設定要求を送信する。 GC1：A は B をコールする。 GC2：B は C にコンサルトコールする。	GC1 および GC2 が通常どおり作成される。	
アプリケーションは TB の GC1 を再開する。	TB（GC2）では、CallCtlTermConnHeldEv TB（GC1）では、CallCtlTermConnTalkingEv CiscoCallFeatureCancelledEv	getCiscoFeatureReason() は、CiscoFeatureReason.REASON_NORMAL を返す。 CiscoCallFeatureCancelledEv.getCall() は、GC1 を返す。 CiscoCallFeatureCancelledEv.getConsultCall() は、GC2 を返す。
アプリケーションは別のコールを作成し、consult() 要求をコール D に送信し、D が応答する。	コンサルト コールは成功し、GC3 が通常どおり作成される。	
使用例 8b		
コンサルト転送/会議：接続された転送/会議をサポートする電話での プライマリ コールを手動で再開してから、別のコンサルト設定要求を送信する。 GC1：A は B をコールする。 GC2：B は C にコンサルトコールする。	GC1 および GC2 が通常どおり作成される。	
ユーザは手動で B の GC1 を再開（SWAP）する。	手動による再開またはスワップで、その電話でコンサルト コールはキャンセルされず、また、アプリケーションが CiscoCallFeatureCancelledEv の取得もしない。	
アプリケーションは別のコールを作成し、consult() 要求をコール D に送信し、D が応答する。	アプリケーションが別のコンサルトを設定しようとすると、それは成功し（GC3 が通常どおり作成され）、既存のコンサルト コールがキャンセルされ、アプリケーションが CiscoCallFeatureCancelledEv を取得する。	CiscoCallFeatureCancelledEv.getCall() は、GC1 を返す。 CiscoCallFeatureCancelledEv.getConsultCall() は、GC2 を返す。

使用例 9		
接続された会議 A（接続された会議を許可する電話）が B をコールし、B が応答し、B が A を保留にする。 B は C をコールし、C が応答する。	GC1 と GC2 のコールが通常どおり作成される。 C1: CallCtlTermConnHeldEv for TB	
B は、[Conference] ハードキーを押し、UI からアクティブ コールを選択し、A→B コールを選択する。	GC2: CallCtlTermConnHeldEv for TB GC3: CiscoConsultCallActiveEv GC3: ConnCreatedEv for B GC3: ConnConnectedEv for B GC3: CallCtlConnInitiatedEv for B GC3: TermConnCreatedEv for TB GC3: TermConnActiveEvent for TB GC3: CallCtlTermConnTalkingEv for TB	
B は [Conference] キーを再度押し、接続された会議を開催する。	GC3: TermConnDroppedEv for TB GC3: CallCtlTermConnDroppedEv for TB GC3: ConnDisconnectedEv for B GC3: CallCtlConnDisconnectedEv for B GC3: CallInvalidEvent GC3: CallObservationEndedEv GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoConferenceEndEv	

使用例 10		
電話からの会議を打診してから、電話で会議をスワップおよび開催する。 A が B にコールし、B が応答する。 B が C に会議を設定し、C が応答する。	GC1 と GC2 のコールが通常どおり作成される。	
B は、[Swap] ソフトキーを押す。	GC2: CallCtlTermConnHeldEv for TB GC1: CallCtlTermConnTalkingEv for TB	getCiscoFeatureReason() は、CiscoFeatureReason.REASON_NORMAL を返す。
A が [Conference] キーを押す。	GC2: CiscoConferenceStartEv GC1: CiscoCallChangedEv GC2: ConnCreatedEv for A GC2: ConnConnectedEv for A GC2: CallCtlConnEstablishedEv for A GC2: TermConnCreatedEv for TA GC2: TermConnActiveEvent for TA GC2: CallCtlTermConnTalkingEv for TA GC1: TermConnDroppedEv for TA GC1: CallCtlTermConnDroppedEv for TA GC1: ConnDisconnectedEv for A GC1: CallCtlConnDisconnectedEv for A GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEv for B GC1: CallCtlConnDisconnectedEv for B GC1: CallInvalidEvent GC1: CallObservationEndedEv GC2: CiscoTransferEndEv	

使用例 11		
電話からの会議を打診してから、電話で会議をスワップおよびキャンセルする。A が B をコールし、B が応答する。 B が C に会議を設定し、C が応答する。	GC1 と GC2 のコールが通常どおり作成される。	
A は Swap キーを押し、UI からアクティブ コールを選択し、A→B コールを選択する。	GC1: CallCtlTermConnTalkingEv for TB GC2: CallCtlTermConnHeldEv for TB	getCiscoFeatureReason() は、CiscoFeatureReason.REASON_NORMAL を返す。
B は、[Cancel] キーを押す。	GC1: CiscoCallFeatureCancelledEv(consultCall = GC2)	CiscoCallFeatureCancelledEv.getCall() は、GC1 を返す。 CiscoCallFeatureCancelledEv.getConsultCall() は、GC2 を返す。
使用例 12		
回線をまたいで接続された会議	一時コール GC3 がある以外は、JAL シナリオと同じ	

使用例 13 手動コンサルトのあと、アプリケーションによって転送を実行する。 GC1：A が B1 をコールする。 GC2：B1 が電話によって手動で C へのコンサルト コールを設定する。 G1.transfer(GC2)	転送時： GC1: CiscoTransferStartEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C **GC1: CiscoCallFeatureCancelledEv** GC1: TermConnDroppedEv for TB GC1: CallCtlTermConnDroppedEv for TB GC1: ConnDisconnectedEvent for B1 GC1: CallCtlConnDisconnectedEv for B1 GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoTransferEndEv	

使用例 14 手動コンサルトのあと、アプリケーションによって会議を開催する。 GC1：A が B1 をコールする。 GC2：B1 が電話によって手動で C へのコンサルト コールを設定する。 G1.conference(GC2)	会議時： **GC1: CiscoCallFeatureCancelledEv** GC1: CiscoConferenceStartEv GC1: CiscoCallFeatureCancelledEv GC1: CiscoCallChangedEv GC1: ConnCreatedEvent for C GC1: ConnConnectedEvent for C GC1: CallCtlConnEstablishedEv for C GC1: TermConnCreatedEvent for TC GC1: TermConnActiveEvent for TC GC1: CallCtlTermConnTalkingEv TC GC2: TermConnDroppedEv for TC GC2: CallCtlTermConnDroppedEv for TC GC2: ConnDisconnectedEvent for C GC2: CallCtlConnDisconnectedEv for C GC2: TermConnDroppedEv for TB GC2: CallCtlTermConnDroppedEv for TB GC2: ConnDisconnectedEvent for B2 GC2: CallCtlConnDisconnectedEv for B2 GC2: CallInvalidEvent GC1: CiscoConferenceEndEv	

任意の通話者のドロップ（Drop Any Party）の使用例

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、FALSE にセットされます。
- Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」は、「never」にセットされます。

シナリオ	操作	結果	CallInfo
使用例 1 アプリケーションは A を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	N.A.
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 2 アプリケーションは C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	N.A.
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C が会議から削除される。	
使用例 3 アプリケーションは、A と C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	N.A.
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 4 アプリケーションは B を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 5 アプリケーションは A、B、および C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、TRUE にセットされます。
- Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」は、「never」にセットされます。

シナリオ	操作	結果	CallInfo
使用例 6 アプリケーションは A を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 7 アプリケーションは C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 8 アプリケーションは、A と C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 9			
アプリケーションは B を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C CallInvalidEv B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 10			
アプリケーションは A、B、および C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、FALSE にセットされます。A と A' は共用回線です。
- Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」は、「never」にセットされます。

シナリオ	操作	結果	コール情報
使用例 11			
アプリケーションは A を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getPartyInfo() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A と B の接続。A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 12			
アプリケーションは A' を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv .A'（「xyz」）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'（「xyz」）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 13			
アプリケーションは、A と A' を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	InvalidStateException がスローされる。	
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A（abc）と A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 14			N.A.
アプリケーションは B を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	
		イベントなし	
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	A'（xyz）が会議から削除される。	
		イベントなし	
		A（abc）が会議から削除される。	
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEv B が会議から切断される。	
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A と B の接続は切断され、A（abc）と A'（xyz）が削除される。B だけが残るため、B も削除され、コールは無効になる。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

使用例 15			
アプリケーションは A、A'、および B を監視している。B が会議コントローラ。A、A'、および B が会議中。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A と B の接続は切断され、A（abc）と A'（xyz）が削除される。B だけが残るため、B も削除され、コールは無効になる。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、TRUE にセットされます。A と A' は共用回線です。
- Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」は、「never」にセットされます。

シナリオ	操作	結果	CallInfo
使用例 16			
アプリケーションは A を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	イベントなし A'（xyz）が会議から切断される。	
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

使用例 17 アプリケーションは A' を監視している。B が会議コントローラ。A、A'、および B が会議中。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	イベントなし A（abc）が会議から切断される。	
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

使用例 18 アプリケーションは、A と A' を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' イベントなし A'（xyz）が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

使用例 19 アプリケーションは B を監視している。B が会議コントローラ。A、A'、および B が会議中。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。 ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid B が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	イベントなし A'（xyz）が会議から切断される。	
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	イベントなし A（abc）が会議から削除される。	
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A（abc）、A'（xyz）および B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

使用例 20 アプリケーションは A、A'、および B を監視している。B が会議コントローラ。A、A'、および B が会議中。	アプリケーションは A の接続で、Connection.getDisplayNames () を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName () と共に、CiscoPartyInfo[] を返す。	N.A.
A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは B の接続で、Connection.disconnect () を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' A'（xyz）が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A（abc）、A'（xyz）および B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

A、B、C が会議中。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「True」が返される。	N.A.
A、B、C が会議中。B が会議から抜ける。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「False」が返される。	N.A.
A、B、B' が会議中。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「True」が返される。	N.A.
A、B、B' が会議中。B' が会議から抜ける。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「False」が返される。	N.A.
A、B、C が会議中。アプリケーションがプロバイダーをオープンし、スナップショット コール イベントを取得する。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「True」が返される。	N.A.
A、B、B' が会議中。アプリケーションがプロバイダーをオープンし、スナップショット コール イベントを取得する。	アプリケーションが CiscoCall.isConferenceCall() を呼び出す。	インターフェイスから「True」が返される。	N.A.

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、FALSE にセットされます。
- コントローラが抜けるときに、Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」がセットされます。

シナリオ	操作	結果	CallInfo
使用例 21			
アプリケーションは A、B、および C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv CallCtlTermConnDroppedEv ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A CallInvalidEV A、B、C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、TRUE にセットされます。

- コントローラが抜けるときに、Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」がセットされます。

シナリオ	操作	結果	CallInfo
使用例 22			
アプリケーションは A、B、および C を監視している。B が会議コントローラ。A、B、および C が会議中。	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは C の接続で Connection.disconnect() を呼び出す。	TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B TermConnDropEv-TC CallCtlTermConnDroppedEv-TC ConnDisconnectedEv-C CallCtlConnDisconnectedEv-C TermConnDropEv-TA CallCtlTermConnDroppedEv-TA ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A、B、C が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。

- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、FALSE にセットされます。
- コントローラが抜けるときに、Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」がセットされます。

シナリオ	操作	結果	CallInfo
使用例 23 アプリケーションは A、A'、および B を監視している。B が会議コントローラ。A、A'、および B が会議中。 A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' A'（xyz）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは B の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
		TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A、A'、および B が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE

	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalidEv A と B の接続は切断され、A（abc）と A'（xyz）が削除される。B だけが残るため、B も削除され、コールは無効になる。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

- JTAPI INI パラメータが有効化され、dropAnyPartyFeature が許可されます。
- Cisco Unified CM サービス パラメータ「Advanced Ad Hoc Conference Enable」は、TRUE にセットされます。
- コントローラが抜けるときに、Cisco Unified CM サービス パラメータ「Drop Ad Hoc Conference」がセットされます。

シナリオ	操作	結果	CallInfo
使用例 24 アプリケーションは A、A'、および B を監視している。B が会議コントローラ。A、A'、および B が会議中。	アプリケーションは A の接続で、Connection.getDisplayNames() を呼び出す。	JTAPI は、「abc」と「xyz」の CiscoPartyInfo.getDisplayName() と共に、CiscoPartyInfo[] を返す。	N.A.

A の表示名は「abc」で、A' の表示名は「xyz」。	アプリケーションは B の接続で、Connection.disconnect () を呼び出す。	TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
		TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A A、A'、および B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_CONFERENCE
	アプリケーションは A の接続で、Connection.disconnect(xyz) を呼び出す。	TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' A'（xyz）が会議から切断される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect(abc) を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA A（abc）が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL
	アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	TermConnDropEv-TA CallCtlTermConnDroppedEv-TA TermConnDropEv-TA’ CallCtlTermConnDroppedEv-TA' ConnDisconnectedEv-A CallCtlConnDisconnectedEv-A TermConnDropEv-TB CallCtlTermConnDroppedEv-TB ConnDisconnectedEv-B CallCtlConnDisconnectedEv-B CallInvalid A（abc）、A'（xyz）および B が会議から削除される。	Cause = CAUSE_NORMAL CiscoFeatureReason = REASON_NORMAL

パーク モニタリング サポート

電話 B：Cisco Unified IP Phone 7900 Series with SIP/SCCP

電話 A：将来のモデル

電話 A'：Cisco Unified IP Phone 7900 Series with SIP/SCCP

パーク DN：P1、P2

電話 C：Cisco Unified IP Phone 7900 Series with SIP/SCCP

パーク モニタリング復帰タイマーと Park Monitoring Forward No Retrieve タイマーには、すべてのデフォルト値が適用されます。

使用例 1：パーク モニタリングの状態

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント/コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 2 ステップ 1 の後、パーク モニタリング復帰タイマーは、設定された時間を過ぎると期限切れになる。	**A** のアドレス オブザーバで、イベントが受信される。 CiscoAddrParkStatusEv **A**	 Cause= CAUSE_NORMAL park state= REMINDER sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 3 ステップ 1 または 2 の後、アプリケーションは端末 C で、パーク解除要求 CiscoTerminal.unpark() を送信する。	**A、B のコール オブザーバで、イベントが受信される。** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** **CiscoAddrParkStatusEv** A	Cause= CAUSE_NORMAL park state= RETRIEVED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 4 上記ステップ 1 または 2 の後、B は端末 B で、CiscoConnection.disconnect() を呼び出すコールをドロップする。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= ABANDONED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 5 A の 未取得時のパークモニタリング転送の接続先が F として設定されることを考慮する。 ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールが F に転送される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Reason= CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID =CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 6 A の未取得時の転送先が A そのものとして設定されることを考慮する。 • ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールがパーク元回線 A に転送される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Reason= CiscoFeatureReason.FORWARD_ NO_RETRIEVE Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID =CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 7 未取得時の転送先が設定されないことを考慮する。 • ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールがパーク元回線 A に転送/復帰される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Reason= CiscoFeatureReason.PARKREMINDER Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 2: 共用回線シナリオ：Cisco Unified IP Phone でパークする

最初のシナリオ：アプリケーションは、A、B、A' に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A/A' が鳴ります。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 2 未取得時の転送先が設定されないことを考慮する。 • パーク モニタリング復帰タイマーと Park Monitoring Forward No Retrieve タイマーが期限切れになることを考慮する。 • コールがパーク元回線 A に転送される / 復帰する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A **注：現在、すべての共用回線はそのまま鳴ります。**	Reason= CiscoFeatureReason.PARKREMINDER Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 3：共用回線シナリオ：Cisco Unified IP Phone 7900 Series with SIP でパークする

最初のシナリオ：アプリケーションは、A、B、A' に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A/A' が鳴ります。A' が応答します。

操作	結果	イベント/コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **注：Cisco Unified IP Phone 7900 Series がパークしているため、新しいイベントは表示されません。**	
ステップ 2 パーク復帰タイマーが期限切れになるのを考慮する。 • コールがパーク元回線 A に復帰する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **注：Cisco Unified IP Phone（将来モデル）を含むすべての共用回線で、電話 A は着信コールを受信します。**	Reason= CiscoFeatureReason.PARKREMINDER

使用例 4: スナップ ショット シナリオの使用例

最初のシナリオ：アプリケーションは、A、B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加していません。B が A をコールします。A が応答します。

操作	結果	イベント/コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	
ステップ 2 ステップ 1 の後、ここでアプリケーションは、A にアドレス オブザーバを追加する。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_SNAPSHOT park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 3a ステップ 2 の後、パーク モニタリング復帰タイマーが期限切れになる。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 3b ステップ 1 の後、アプリケーションは端末 C で、パーク解除要求 CiscoTerminal.unpark() を送信する。	**A、B のコール オブザーバで、イベントが受信される。** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
ステップ 4 ステップ 3 の後、ここでアプリケーションは、A にアドレス オブザーバを追加する。	**コールがすでに取得されているため、park state=RETRIEVED の新しいアドレスのイベント は A で受信されない。**	

使用例 5：パーク DN が監視されている

最初のシナリオ：アプリケーションは、A、B に、コール オブザーバを追加しています。アプリケーションは、パーク DN P1 を監視するため、プロバイダーで registerFeature() API を呼び出します。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント/コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**プロバイダー オブザーバ Prov1 で、イベントが受信される。** CiscoProvCallParkEv **A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 6：パークされているコールのクエリ番号

最初のシナリオ：アプリケーションは、A、B、C に、コール オブザーバを追加しています。

操作	結果	イベント / コール情報
ステップ 1 B が A をコールする。A が応答する。アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	
ステップ 2 C が A をコールする。A が応答する。アプリケーションは、A での 2 度目のコールによる A の接続で、CiscoConnection.park() を呼び出す。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
ステップ 3 アプリケーションが、CiscoAddress.getAddressCallInfo（Term A）を呼び出す。 アプリケーションが、CiscoAddressCallInfo.getNumParkedCalls() を呼び出す。	パークされているコールの数の情報を含む CiscoAddressCallInfo が返される。 getNumParkedCalls() は、2 を返す。	

使用例 7：フィルタの有効化または無効化

最初のシナリオ：アプリケーションは、A、B に、コール オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 最初に、フィルタが無効化される。 • アプリケーションが A に AddressObserver を追加する。 • アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 • パーク復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** フィルタが無効化されているため、イベントは受信されない。	
ステップ 2 ステップ 1 の後、アプリケーションは setCiscoAddrParkStatusEvFilter(true) でフィルタを有効化し、CiscoAddress.setFilter(CiscoAddrEvFilter) を呼び出して、イベントを受信する。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_SNAPSSHOT park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 8：フィルタの有効化または無効化

最初のシナリオ：電話 B から A をコールします。A が応答します（コール オブザーバは追加されません）。

操作	結果	イベント / コール情報
ステップ 1 最初に、フィルタが有効化される。 • アプリケーションが A に AddressObserver を追加する。 • アプリケーションはここで、電話 A から直接パークを呼び出す。 • パーク復帰タイマーが開始する。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 9：フィルタの有効化または無効化

最初のシナリオ：電話 B から A をコールします。A が応答します（コール オブザーバは追加されません）。

操作	結果	イベント / コール情報
ステップ 1 最初に、フィルタが無効化される。 • アプリケーションが A に AddressObserver を追加する。 • アプリケーションはここで、電話 A から直接パークを呼び出す。 • パーク復帰タイマーが開始する。 • アプリケーションはここでフィルタを有効化し、CiscoAddress.setFilter(CiscoAddrEvFilter) を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが無効化されているため、イベントはまだ受信されない。 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A	Cause= CAUSE_SNAPSHOT park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 2 パーク リマインダ タイマーが期限切れになる。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A	Cause= CAUSE_NORMAL park state= REMINDER sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

使用例 10：フィルタの有効化または無効化

最初のシナリオ：アプリケーションは、A、B に、コール オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 最初に、すべてのフィルタが、CiscoAddEvFilter で無効化される。 • アプリケーションが A に AddressObserver を追加する。 • アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 • パーク復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** フィルタが無効化されているため、イベントは受信されない。	
ステップ 2 ステップ 1 の後、アプリケーションは setCiscoAddrParkStatusEvFilter(true) を呼び出すが、CiscoAddress.setFilter(CiscoAddrEvFilter) は呼び出さない。	**A のアドレス オブザーバで、イベントが受信される。** アドレス フィルタがセットされていないため、イベントは受信されない。	
ステップ 3 ここで、アプリケーションは CiscoAddress で、setFilter(CiscoAddrEvFilter) を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A	Cause= CAUSE_SNAPSSHOT park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

パーク モニタリングのその他の使用例

電話 B：Cisco Unified IP Phone 7900 Series with SIP/SCCP

電話 A：将来のモデル

電話 A'：Cisco Unified IP Phone 7900 Series with SIP/SCCP

パーク DN：P1、P2

電話 C：Cisco Unified IP Phone 7900 Series with SIP/SCCP

パーク モニタリング復帰タイマーと Park Monitoring Forward No Retrieve タイマーには、すべてのデフォルト値が適用されます。

1. 最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。フィルタの値が setCiscoAddrParkStatusEvFilter() によって「true」にセットされます。

操作	結果	イベント / コール情報
ステップ 1 • アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 • パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	 Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 2 ステップ 1 の後、パーク モニタリング復帰タイマーは、設定された時間を過ぎると期限切れになる。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	 Cause= CAUSE_NORMAL park state= REMINDER sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 3 ステップ 1 または 2 の後、アプリケーションは端末 C で、パーク解除要求 CiscoTerminal.unpark() を送信する。	**A、B のコール オブザーバで、イベントが受信される。** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= RETRIEVED sub ID =1234 CiscoCallID =CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 4 上記ステップ 1 または 2 の後、B は端末 B で、CiscoConnection.disconnect() を呼び出すコールをドロップ オフする。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 CallInvalidEv **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	 Cause= CAUSE_NORMAL park state= ABANDONED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 5 A の未取得時のパークモニタリング転送の接続先が F として設定されることを考慮する。 • ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールが F に転送される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv F GC1 ConnInProgressEv F GC1 CallCtlConnOfferedEv F GC1 ConnAlertingEv F GC1 CallCtlConnAlertingEv F GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	 Reason= CiscoFeatureReason.FORWARD_NO_RETRIEVE Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID =CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 6 A の未取得時の転送先が A そのものとして設定されることを考慮する。 • ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールがパーク元回線 A に転送される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Reason= FORWARD_NO_RETRIEVE Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID =CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 7 未取得時の転送先が設定されないことを考慮する。 • ステップ 2 の後、Park Monitoring Forward-No-retrieve タイマーが開始する。 • Park Monitoring Forward-No-retrieve タイマーが期限切れになる。 • コールがパーク元回線 A に転送 / 復帰される。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Reason= PARKREMINDER Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

2. 最初のシナリオ：アプリケーションは、A、B、A' に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A/A' が鳴ります。A が応答します。フィルタの値が setCiscoAddrParkStatusEvFilter() によって「true」にセットされます。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 TermConnDroppedEv TA' GC1 CallCtlTermConnDroppedEv TA' GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

操作	結果	イベント / コール情報
ステップ 2 未取得時の転送先が設定されないことを考慮する。 • パーク モニタリング復帰タイマーと Park Monitoring Forward No Retrieve タイマーが期限切れになることを考慮する。 • コールがパーク元回線 A に転送される / 復帰する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 ConnCreatedEv A GC1 ConnInProgressEv A GC1 CallCtlConnOfferedEv A GC1 ConnAlertingEv A GC1 CallCtlConnAlertingEv A GC1 TermConnCreatedEv TA GC1 TermConnRingingEv TA GC1 CallCtlTermConnRingingEvImpl TA GC1 ConnInProgressEv A GC1 ConnAlertingEv A GC1 TermConnCreatedEv TA' GC1 TermConnRingingEv TA' GC1 CallCtlTermConnRingingEvImpl TA' GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1	
	A のアドレス オブザーバで、イベントが受信される。 CiscoAddrParkStatusEv **A** **注：現在、すべての共用回線はそのまま鳴ります。**	Cause= CAUSE_NORMAL park state= FORWARDED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

3. 最初のシナリオ：アプリケーションは、A、B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加していません。B が A をコールします。A が応答します。フィルタの値が setCiscoAddrParkStatusEvFilter() によって「true」にセットされます。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	

ステップ 2 ステップ 1 の後、ここでアプリケーションは、A にアドレス オブザーバを追加する。	**A** のアドレス オブザーバで、イベントが受信される。 CiscoAddrParkStatusEv **A**	Cause= CAUSE_SNAPSHOT park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA
ステップ 3a ステップ 2 の後、パーク モニタリング復帰タイマーが期限切れになる。	**A** のアドレス オブザーバで、イベントが受信される。 CiscoAddrParkStatusEv **A**	Cause= CAUSE_NORMAL park state= REMINDER sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 3b ステップ 1 の後、アプリケーションは端末 C で、パーク解除要求 CiscoTerminal.unpark() を送信する。	**A、B のコール オブザーバで、イベントが受信される。** GC2 CallActiveEv GC2 ConnCreatedEv C GC2 ConnConnectedEv C GC2 CallCtlConnInitiatedEv C GC2 TermConnCreatedEv TC GC2 TermConnActiveEv TC GC2 CallCtlTermConnTalkingEv TC GC2 CallCtlConnDialingEv C GC2 CallCtlConnEstablishedEv C GC2 ConnCreatedEv P1 GC2 ConnInProgressEv P1 GC2 CallCtlConnOfferedEv P1 GC1 CiscoCallChangedEv GC2 ConnCreatedEv B GC2 ConnConnectedEv B GC2 CallCtlConnEstablishedEv B GC2 TermConnCreatedEv TB GC2 TermConnActiveEv TB GC2 CallCtlTermConnTalkingEv TB GC2 ConnConnectedEv P1 GC2 CallCtlConnEstablishedEv P1 GC1 ConnDisconnectedEv P1 GC1 CallCtlConnDisconnectedEv P1 GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B C1 CallInvalidEv GC2 ConnDisconnectedEv P1 GC2 CallCtlConnDisconnectedEv P1	
ステップ 4 ステップ 3 の後、ここでアプリケーションは、A にアドレス オブザーバを追加する。	コールがすでに取得されているため、park state=RETRIEVED の新しいアドレスのイベント は A で受信されない。	

4. 最初のシナリオ：アプリケーションは、A、B、C に、コール オブザーバを追加しています。フィルタの値が setCiscoAddrParkStatusEvFilter() によって「true」にセットされます。

操作	結果	イベント / コール情報
ステップ 1 B が A をコールする。A が応答する。アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1	
ステップ 2 C が A をコールする。A が応答する。アプリケーションは、A での 2 度目のコールによる A の接続で、CiscoConnection.park() を呼び出す。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P2 GC1 ConnInProgressEv P2 GC1 CallCtlConnQueuedEv P2	
ステップ 3 アプリケーションが、CiscoAddress.getAddressCallInfo（Term A）を呼び出す。 アプリケーションが、CiscoAddressCallInfo.getNumParkedCalls() を呼び出す。	パークされているコールの数の情報を含む CiscoAddressCallInfo が返される。 getNumParkedCalls() は、2 を返す。	

5. イベント通知を制御するアドレス イベント フィルタを確認する使用例：フィルタの値は setCiscoAddrParkStatusEvFilter() で「false」にセットされます。これもデフォルト値です。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 デフォルトでは、アドレス イベント フィルタの値は false。アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** フィルタの値が false であるため、イベント通知はない。	

6. イベント通知を制御するアドレス イベント フィルタを確認する使用例。フィルタの値は、setCiscoAddrParkStatusEvFilter() で「true」にセットされています。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、CiscoAddrEvFilter.setCiscoAddrParkStatusEvFilter(true) でフィルタを有効化する。アプリケーションは、A の接続で、CiscoConnection.park() を呼び出す。 パーク モニタリング復帰タイマーが開始する。	**A、B のコール オブザーバで、イベントが受信される。** GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnCreatedEv P1 GC1 ConnInProgressEv P1 GC1 CallCtlConnQueuedEv P1 **A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A	Cause= CAUSE_NORMAL park state= PARKED sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

ステップ 2 上のステップ 1 の後、アプリケーションは、CiscoAddrEvFilter.setCiscoAddrParkStatusEvFilter(false) でフィルタを無効化する。 パーク モニタリング復帰タイマーが期限切れになるのを考慮する。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが無効化されているため、イベント通知はない。	
ステップ 3 上のステップ 2 の後、アプリケーションは、CiscoAddrEvFilter.setCiscoAddrParkStatusEvFilter(true) でフィルタを有効化する。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrParkStatusEv A	Cause= CAUSE_SNAPSHOT park state= REMINDER sub ID =1234 CiscoCallID = CiscoCallID for GC1 park DN =P1 parked party=B terminal= TA

7. イベント CiscoAddrPArkrStatusEv にセットされたフィルタの値を確認する使用例。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、CiscoAddrEvFilter.setCiscoAddrParkStatusEvFilter(false) でフィルタを無効化する。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrParkStatusEvFilter() を呼び出す。	アプリケーションは、ブール値「false」を受信する。	
ステップ 2 アプリケーションは、CiscoAddrEvFilter.setCiscoAddrParkStatusEvFilter(true) でフィルタ値を有効化する。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrParkStatusEvFilter を呼び出す。	アプリケーションは、ブール値「true」を受信する。	

8. インターコム機能（ターゲット DN とインターコム ターゲット ラベルの一方または両方）が変更されていないときに、CiscoAddrIntercomInfoChangedEv の通知およびイベントのフィルタの値を確認する使用例。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A をコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、CiscoAddrEvFilter.setAddrIntercomInfoChangedEvFilter(false) で、フィルタの値を「false」にセットする。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoChangedEvFilter() を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが有効化されているため、アドレス通知はない。 アプリケーションは、ブール値「false」を受信する。	
ステップ 2 アプリケーションは、CiscoAddrEvFilter.setCiscoAddrIntercomInfoChangedEvFilter(true) でフィルタの値を有効化する。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoChangedEvFilter を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** インターコム機能が変更されていないため、イベントは受信されない。 アプリケーションは、ブール値「true」を受信する。	

9. インターコム機能（ターゲット DN とインターコム ターゲット ラベルの一方または両方）が変更されているときに、CiscoAddrIntercomInfoChangedEv の通知およびイベントのフィルタの値を確認する使用例。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A にコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 アプリケーションは、CiscoAddrEvFilter.setAddrIntercomInfoChangedEvFilter(false) で、フィルタの値を「false」にセットする。 アプリケーションは、インターコム アドレス A で、CiscoIntercomAddress.setIntercomTarget() を発行する。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoChangedEvFilter() を呼び出す。	A のアドレス オブザーバで、イベントが受信される。 フィルタが有効化されているため、アドレス通知はない。 アプリケーションは、ブール値「false」を受信する。	
ステップ 2 アプリケーションは、CiscoAddrEvFilter.setCiscoAddrIntercomInfoChangedEvFilter(true) でフィルタの値を有効化する。 アプリケーションは、インターコム アドレス A で、CiscoIntercomAddress.setIntercomTarget() を発行する。 アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoChangedEvFilter を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrIntercomInfoChangedEv **A** アプリケーションは、ブール値「true」を受信する。	

10. CiscoAddrIntercomInfoRestorationFailedEv の通知と、このイベントのフィルタの値を確認する使用例。

    最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A にコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 • アプリケーションは、CiscoAddrEvFilter.setCiscoAddrIntercomInfoRestorationEvFilter(false) で、フィルタの値を「false」にセットする。 • アプリケーションは、インターコム アドレス A でインターコム ターゲット DN およびラベルを設定している。ここで、CTIManager がアウトオブ サービスになり、JTAPI は別の CTIManager ノードにフェールオーバーする。インターコム アドレス A がイン サービスに戻った後、JTAPI は、インターコム ターゲット DN、ラベルおよびユニコードラベルを、設定値に復旧する。ただし、他のアプリケーションがすでにターゲット DN を設定しているという競合状態が原因で、JTAPI は CTI から失敗の応答を受信する。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoRestorationEvFilter() を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが有効化されているため、通知はない。 アプリケーションは、ブール値「false」を受信する。	
ステップ 2 • アプリケーションは、API CiscoAddrEvFilter.setCiscoAddrIntercomInfoRestorationEvFilter(true) でフィルタを有効化する。 • アプリケーションは、インターコム アドレス A でインターコム ターゲット DN およびラベルを設定している。ここで、CTIManager がアウトオブ サービスになり、JTAPI は別の CTIManager ノードにフェールオーバーする。インターコム アドレス A がイン サービスに戻った後、JTAPI は、インターコム ターゲット DN、ラベルおよびユニコードラベルを、設定値に復旧する。ただし、他のアプリケーションがすでにターゲット DN を設定しているという競合状態が原因で、JTAPI は CTI から失敗の応答を受信する。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrIntercomInfoRestorationEvFilter() を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrIntercomInfoRestorationFailedEv **A** アプリケーションは、ブール値「true」を受信する。	

11. CiscoAddrRecordingConfigChangedEv の通知と、このイベントのフィルタの値を確認する使用例。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリ

ケーションは、A に、アドレス オブザーバを追加しています。B が A にコールします。A が応答します。

録音プロファイルの設定は、変更されていません。

操作	結果	イベント/コール情報
ステップ 1 • アプリケーションは、CiscoAddrEvFilter.setCiscoAddrRecordingConfigChangedEvFilter で、フィルタの値を「false」にセットする。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrRecordingConfigChangedEvFilter() を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが有効化されているため、アドレス通知はない。 アプリケーションは、ブール値「false」を受信する。	
ステップ 2 • アプリケーションは、CiscoAddrEvFilter.setCiscoAddrRecordingConfigChangedEvFilter(true) でフィルタ値を有効化する。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrRecordingConfigChangedEvFilter を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** 録音設定が変更されていないため、イベントはない。 アプリケーションは、ブール値「true」を受信する。	

12. CiscoAddrRecordingConfigChangedEv の通知と、このイベントのフィルタの値を確認する使用例。

最初のシナリオ：アプリケーションは、A と B に、コール オブザーバを追加しています。アプリケーションは、A に、アドレス オブザーバを追加しています。B が A にコールします。A が応答します。

操作	結果	イベント / コール情報
ステップ 1 • アプリケーションは、CiscoAddrEvFilter.setCiscoAddrRecordingConfigChangedEvFilter (false) で、フィルタの値を「false」にセットする。 ユーザは、管理ページで、録音プロファイルの設定を変更する。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrRecordingConfigChangedEvFilter() を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** フィルタが有効化されているため、アドレス通知はない。 アプリケーションは、ブール値「false」を受信する。	
ステップ 2 • アプリケーションは、CiscoAddrEvFilter.setCiscoAddrRecordingConfigChangedEvFilter(true) でフィルタ値を有効化する。 ユーザは、管理ページで、録音プロファイルの設定を変更する。 • アプリケーションが、CiscoAddrEvFilter で API getCiscoAddrRecordingConfigChangedEvFilter を呼び出す。	**A のアドレス オブザーバで、イベントが受信される。** CiscoAddrRecordingConfigChangedEv **A** アプリケーションは、ブール値「true」を受信する。	

DPark 関連の使用例

最初のセットアップは、次のようになっています。

- アプリケーションは、B および A で、コール オブザーバを追加済みです。

- ユーザには、設定済みの DPark DN D があります。

- B は、Cisco Unified IP Phone の将来モデルです。

- A が B にコールします。B が、GCID GC1 で応答します。

コール シナリオ	予想される動作
Cisco Unified IP Phone から Assisted DPark： • Cisco Unified IP Phone の電話 B（A とのアクティブ コール中）が、事前設定済みの DPark BLF ボタンを押す。 • パークされている通話者 A は、D に接続され、MoH を聞く。	A（パークされている通話者）は、D に接続され、次のイベントが受信される。 **B、A のコール オブザーバで、イベントが受信される。** GC1 CallCtlTermConnHeldEv TB (CiscoFeatureReason.REASON_DPARK_CALLPARK) GC1 CiscoTermConnSelectChangedEv TB GC1 ConnUnknownEv B GC1 CallCtlConnUnknownEv B GC1 TermConnUnknownEv TB (CiscoFeatureReason.REASON_REFER)) GC1 ConnCreatedEv D GC1 ConnInProgressEv D GC1 CallCtlConnQueuedEv D (CiscoFeatureReason.REASON_REFER)) GC1 TermConnDroppedEv TB GC1 CallCtlTermConnDroppedEv TB GC1 ConnDisconnectedEv B GC1 CallCtlConnDisconnectedEv B(CiscoFeatureReason.REASON_REFER))
Cisco Unified IP Phone からの DPark • Cisco Unified IP Phone の電話 B（A とのアクティブ コール中）が、[Transfer] ソフトキーを押す。 • パークされている通話者 A は、保留になり、パーク元 B は、D にダイヤルする。 • パーク元 B は、D に接続され、MoH を聞く。 • パーク元 B は、[Transfer] ソフトキーを再度押し、パークされている通話者 A から Dpark コード D への転送を実行する。 • パークされている通話者 A は、D に接続される。	動作に変化はなし。すべてのイベント / 原因は、現在と変わらない。
JTAPI API からの DPark • アプリケーションは、consult() API を使用して、宛先アドレス DPark DN D で B からのコンサルト コールを要求する。たとえば、このコールに GCID–GC2 があるとする。 • アプリケーションは、transfer() API GC1.transfer(GC2) を使用して、転送を実行する。 • 転送が実行されると、A は DPark DN に接続される。	動作に変化はなし。すべてのイベント / 原因は、現在と変わらない。

コール シナリオ	予想される動作
JTAPI API からのパーク解除 • アプリケーションが C を監視していることを考慮する。 • ステップ 3 の後、アプリケーションは、connect() API を使用し、プレフィクス コードに DPark コードを続けたものを宛先アドレスとして、パーク解除要求を発行する。 • A が C に接続されている。	動作に変化はなし。すべてのイベント/原因は、現在と変わらない。
JTAPI API による DPark DN へのリダイレクト • B は、redirect() API を使用し、D をリダイレクト宛先として、DPark コード D にリダイレクトする。	動作に変化はなし。B は、DPark DN にリダイレクトされるが、パーク操作はない。

論理パーティション設定機能の使用例

Logical Partition（LP; 論理パーティション）制限クラスタからリダイレクトします。

端末 TA は、アドレス A で設定され、論理パーティション制限で設定されるクラスタに登録されます。端末 TX は、LP 設定のないクラスタに設定されるアドレス X で登録されます。

操作	結果
X が A をコールする。A は、ローカルの PSTN 番号にコールをリダイレクトする。	PlatformException がスローされ、要求がリダイレクトされる。 例外の getErrorCode() が、CiscoJtapiException を返す。REDIRECT_CALL_PARTITIONING_POLICY
A が X（GC1）をコールし、X のローカルの PSTN 番号にコールをリダイレクトする。	元のクラスタは、そのコールが PSTN にリダイレクトされることを認識し、コールを切断する。 **A のコール オブザーバに配信されるイベント** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv A GC1 CallCtlTermConnDroppedEv TA GC1 ConnDisconnectedEv A GC1 CallCtlConnDisconnectedEv A GC1 ConnDisconnected X GC1 CallCtlConnDisconnectedEv X GC1 CallInvalidEv

コール転送：「許可されない」ポリシーで、コール先のアドレスから GeoLocation にある PSTM へ、すべて転送されるコール。

操作	結果
A で、ローカル PSTN への setForward を試みる。 アプリケーションは、X を監視している。 • GC1.connect() を使用して、X が A をコールする。	Connect() API は成功するが、A 側の制限により、コールがドロップされる。 **X のコール オブザーバに配信されるイベント** GC1 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1 CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED GC1: GC1 TermConnDroppedEv X GC1 CallCtlTermConnDroppedEv TX GC1 ConnDisconnectedEv X GC1 CallCtlConnDisconnectedEv X GC1 ConnDisconnected A GC1 CallCtlConnDisconnectedEv A GC1 CallInvalidEv

コール転送：GeoLocation にあるコントローラによる、「許可されない」ポリシーでの、別の地域から PSTN へのコールの転送。

操作	結果
X が A にコールし、A が PSTN 番号にコンサルト コールする。 アプリケーションは、A を監視している。 • A が転送を実行する。	プラットフォーム例外が、transfer() 要求にスローされる。 getErrorCode() は、CiscoJtapiException.TRANSFERFAILED を返す。

電話会議：GeoLocation にあるコントローラによる、「許可されない」ポリシーでの、別の地域から PSTN への電話会議の開催。

操作	結果
X が A にコールし、A が PSTN 番号にコンサルト コールする。 アプリケーションは、A を監視している。 • A が会議を実行する。	プラットフォーム例外が、conference() 要求にスローされる。 getErrorCode() は、CiscoJtapiException.CTIERR_FEATURE_NOT_AVAILABLE を返す。

パーク / パーク解除のコール：PSTN コールをパークおよびパーク解除します。

A と B は同じクラスタ内にありますが、LP 制限で、別の地域に設定されています。PSTN は、B と同じ地域です。

操作	結果
PSTN 番号が A にコールし、A が応答してコールをパークする。	
アプリケーションが A と B を監視している。 • B が、unpark() API を使用してコールをパーク解除する。	コールは失敗する。 ConnFailedEv A CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED CallCtlConnFailedEv CAUSE: CiscoCallEv.CAUSE_SERVNOTAVAILUNSPECIFIED

共用回線

TermA と TermA' は、同じクラスタ内にありますが、LP 制限により、別の地域で設定されます。PSTN は、TermA と同じ地域です。

操作	結果
PSTN 番号が A にコールする。TermA だけが鳴る。	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

異なる地域の共用回線でのコール パーク制限

TermA と TermA' は、同じクラスタ内にありますが、LP 制限により、別の地域で設定されます。PSTN は、TermA と同じ地域です。

操作	結果
PSTN 番号が A にコールし、TermA が応答してコールをパークする。	
タイム アウト コールが、TermA' ではなく TermA で提供される。	GC1: CallActiveEv GC1: ConnAlertingEv A GC1: TermConnRingingEv TermA GC1: CallCtlTermConnRingingEv TermA

ComponentUpdater 拡張の使用例

操作	結果
アプリケーションが、ComponentUpdater(null) をコールする。	同じディレクトリ内に、updater.log が作成される。
アプリケーションが、ComponentUpdater（「C:¥¥temp¥¥」）をコールする。	C:¥temp に updater.log が作成される。
アプリケーションが、ComponentUpdater('readonlydirectory') をコールする。 アプリケーションには、Readonlydirectory への書き込み権限がない。	ログは作成されない。

IPv6 のサポート

getIPAddressingMode() の使用例

操作	結果
ステップ 1 Unified CM の管理ページもある、端末 A の IP アドレッシング モードが、IPv4 にセットされる。 アプリケーションが端末 A で、CiscoTerminal.getIPAddressingMode() を呼び出す。	getIPAddressingMode() が 0 を返す。
ステップ 2 ステップ 1 の後、IP アドレッシング モードは IPv4 から IPv6 に変わる。ユーザにデバイスのリセットを求めるメッセージが表示される。 ここで、アプリケーションが端末 A で、CiscoTerminal.getIPAddressingMode() を呼び出す。	getIPAddressingMode() が 1 を返す。 （指定されたユーザがデバイスをリセットした）

Cisco Unified IP Phone 6900 シリーズのサポート

シナリオ / 説明	アプリケーションへのイベント
アプリケーションは CTIManager に接続する。 • TermA は、Cisco Unified IP Phone 6921。 • TermB は、ロール オーバー モードの Cisco Unified IP Phone 7931。 管理者は、ここで新しいユーザ ロールを有効化する。	CiscoProviderCapabilityChangedEv : CiscoProvider.canObserverTerminalsWithRoleOver() は、true を返す。 CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() は、true を返す。 プロバイダー オブザーバへのイベント： CiscoTermActivatedEv TermA CiscoTermActivatedEv TermB
管理者は、新しいユーザ ロールを削除する。	CiscoProviderCapabilityChangedEv : CiscoProvider.canObserverlTerminalsWithRoleOver() は、false を返す。 CiscoProviderCapabilityChangedEv .hasObserverTerminalsWithRoleOverChanged() は、true を返す。 プロバイダー オブザーバへのイベント： CiscoTermRestrictedEv TermA CiscoTermRestrictedEv TermB
Term A は、Cisco Unfied IP 6900 シリーズの電話。アプリケーションは、新規権限を有効化しない。TermA は、アプリケーション制御リストにある。 アプリケーションが TermA にオブザーバを追加する。	PlatformException がスローされる。getErrorCode() は、CiscoJtapiException.CTIERR_DEVICE_RESTRICTED を返す。
コール シナリオ：	

シナリオ / 説明	アプリケーションへのイベント
TermA は、アドレス A、A:P1、A:P2（P1 および P2 は 2 つのパーティション）で設定される。 アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。	端末オブザーバに配信されるイベント GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA
TermA は、コールを同じ DN にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが、端末に callObserver を追加する。X が A にコールし、アプリケーションがコール（GC1）に応答する。	GC1: CallCtlTermConnHeldEv TermA
アプリケーションが、Y（GC2）にコンサルト要求を発行する。A:P1 にコールが作成される。	GC2: CallActiveEv GC2: ConnCreatedEv **A:P1** GC2: ConnConnectedEv **A:P1** GC2: CallCtlConnInitiatedEv **A:P1**
着信コールへのロールオーバーなし： TermA は、アドレス A、A:P1、A:P2（P1 および P2 は 2 つのパーティション）で設定される。 アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。	端末オブザーバに配信されるイベント GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA
TermA は、コールを同じ DN にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが、端末に callObserver を追加する。X が A にコールし、アプリケーションがコール（GC1）に応答する。 connect API を使用して、アプリケーションが B から A にコールする。	GC2: CallActiveEv GC2: ConnCreatedEv **B** GC2: ConnConnectedEv **B** GC2: CallCtlConnInitiatedEv **B** GC2: TermConnCreatedEv TernB GC2: CallCtlConnDialingEv B GC2: CallCtlConnEstablishedEv B GC2: ConnFailedEv B. getCiscoCause() は、CiscoCallEv.CAUSE_USERBUSY を返す。

シナリオ / 説明	アプリケーションへのイベント
転送および会議のロール オーバー（consult()）のみ： TermA は、アドレス A、A:P1、A:P2（P1 および P2 は 2 つのパーティション）で設定される。 アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。 TermA は、コールを同じ DN にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが、端末に callObserver を追加する。X が A にコールし、アプリケーションがコール（GC1）に応答する。 アプリケーションは、アドレス A から Y に connect() API を呼び出す。同様の例外が、unPark() 要求、startMonitor() 要求に表示される。	**端末オブザーバに配信されるイベント** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException がスローされる。getErrorCode() は、CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED を返す。
1 アドレスだけに callObserver がある： TermA は、アドレス A、A:P1、A:P2（P1 および P2 は 2 つのパーティション）で設定される。 アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。 TermA は、コールを同じ DN にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが、アドレス A だけに callObserver を追加する。 X が A にコールし、A がコールに応答する。 アプリケーションが consult() API を使用して、Y にコンサルト コールする。電話呼び出しの結果、A:P1 にコンサルト コールが 作成される。	**A の CallObserver に配信されるイベント** GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA PlatformException がスローされる。getErrorDescription() は、「No callobserver on address A:P1」を返す。getErrorCode() は、CiscoJtapiException.ASSOCIATED_LINE_NOT_OPEN を返す。

シナリオ / 説明	アプリケーションへのイベント
任意の回線へのロール オーバー	
ロール オーバーでは、同じ DN のアドレスにプリファレンスが与えられる。同じ DN のアドレスが利用できれば、コンサルト コールのロール オーバーのために選択される。	**端末オブザーバに配信されるイベント**
TermA は、アドレス A、**B**、A:P1（P1 はパーティション）で設定される。アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。TermA は、コールを任意の回線にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが TermA に callObserver を追加する。	GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA
X が A にコールし、アプリケーションがコールに応答する。	GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv A:P1 GC2: ConnConnectedEv A:P1 GC2: CallCtlConnInitiatedEv A:P1
アプリケーションが Y にコンサルト コールする。コンサルト コールが、回線 3 に作成される。	

シナリオ / 説明	アプリケーションへのイベント
任意の回線へのロール オーバー（同じ DN は他のコールが使用中）：	
TermA は、アドレス A、**B**、A:P1（P1 はパーティション）で設定される。アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。TermA は、コールを任意の回線にロール オーバーするよう設定される。最大コール数とビジー トリガーは 1 にセットされる。アプリケーションが TermA に callObserver を追加する。	**端末オブザーバに配信されるイベント**
GC1: A:P1 が Z にコールし、A:P1 がコールを保留する。	GC2: ConnConnectedEv **A** GC2: CallCtlConnEstablishedEv **A** GC2: CallCtlTermConnTalkingEv TermA
GC2:X が A にコールし、アプリケーションがコールに応答する。	GC2: CallCtlTermConnHeldEv TermA GC3: CallActiveEv GC3: ConnCreatedEv B GC3: ConnConnectedEv B GC3: CallCtlConnInitiatedEv B
アプリケーションが、GC2 を Y（GC3）にコンサルトコールする。	... GC3: ConnCreatedEv Y .. GC3: CallCtlConnAlertingEv Y .. GC3: ConnConnectedEv Y GC3: CallCtlConnEstablishedEv Y
	CiscoTransferStartEv getTransferControllerAddress() は A を返す。 ... CiscoTransferEndEv
アプリケーションが転送を実行する。	

シナリオ / 説明	アプリケーションへのイベント
最大コール数 > 1 : TermA は、アドレス A、A:P1（P1 はパーティション）で設定される。アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。TermA は、コールを任意の回線にロール オーバーするよう設定される。A で、最大コール数とビジー トリガーは 3 および 2 にセットされる。アプリケーションが TermA に callObserver を追加する。 X が A にコールし、A が応答する。 アプリケーションが Y にコンサルト コールする。 コンサルト コールが A に設定される（同じ回線）。	**端末オブザーバに配信されるイベント** ... ... GC1: ConnConnectedEv **A** GC1: CallCtlConnEstablishedEv **A** GC1: CallCtlTermConnTalkingEv TermA GC1: CallCtlTermConnHeldEv TermA GC2: CallActiveEv GC2: ConnCreatedEv **A** GC2: ConnConnectedEv **A** GC2: CallCtlConnInitiatedEv **A**
TermA は、アドレス A、A:P1（P1 はパーティション）で設定される。アプリケーションは、新規権限「Standard CTI Allow Control of Phones supporting roll over mode」を有効化する。TermA は、コールを任意の回線にロール オーバーするよう設定される。A および A:P1 で、最大コール数とビジー トリガーは 1/1 にセットされる。アプリケーションが TermA に callObserver を追加する。 A1 が X にコールする。X がコールに応答する：GC1 Y が A にコールする。A がコールに応答し、Z にコンサルト コールする。	PlatformException がスローされる。getErrorCode() は、CiscoJtapiException.CTIERR_MAXCALL_LIMIT_REACHED を返す。