Cisco Unified JTAPI 開発者ガイド for Cisco Unified Communications Manager リリース 7.1(2)

Cisco Unified JTAPI Developers Guide for Cisco Unified Communications Manager Release 7.1(2)

Text Part Number: OL-18531-01-J

【注意】シスコ製品をご使用になる前に、安全上の注意（www.cisco.com/jp/go/safety_warning/）をご確認ください。

本書は、米国シスコシステムズ発行ドキュメントの参考和訳です。米国サイト掲載ドキュメントとの差異が生じる場合があるため、正式な内容については米国サイトのドキュメントを参照ください。また、契約等の記述については、弊社販売パートナー、または、弊社担当者にご確認ください。

Cisco Unified JTAPI 開発者ガイド for Cisco Unified Communications Manager Release 7.1(2)

CONTENTS

はじめに

この章では、本書の目的、対象読者、本書の構成、および本書での説明の表記方法について説明します。次のような構成になっています。

目的

このドキュメントでは、Cisco Unified Communications Manager（以前の Cisco Unified CallManager）の、Java Telephony Application Programming Interface（JTAPI）について説明します。Cisco Unified JTAPI では、プログラミング インターフェイスで複数の実装をサポートできます。このドキュメントでは、拡張、クラスおよびインターフェイスを含む、Cisco Unified JTAPI の基本的な概念について概説します。

シスコでは、Cisco Unified Communications Manager の JTAPI を実装するにあたり、JTAPI 仕様 v1.2 に準拠しつつ、Cisco Unified Communications Manager の高度な機能を備えた JTAPI を強化するための拡張を提供しています。

シスコでは、Cisco Unified Communications Manager および Cisco Unified JTAPI 実装の新バージョンをリリースする際、API 拡張の安定性および信頼性を維持します。また、Cisco Unified Communication Manager の新機能が追加された際には、新たな拡張を提供します。

対象読者

このドキュメントは、本書で説明される API の機能を拡張するアプリケーションの開発者を対象としています。

このドキュメントでは、Java 言語および Sun JTAPI v 1.2 仕様の知識があることを前提として書かれています。また、次の領域の知識や経験も必要としています。

- Extensible Markup Language（XML）
- Hypertext Markup Language（HTML）
- Hypertext Transport Protocol（HTTP; ハイパーテキスト転送プロトコル）
- ソケット プログラミング
- TCP/IP プロトコル
- Web Service Definition Language（WSDL）1.1
- Secure Sockets Layer（SSL）

さらに、Cisco Unified Communications Manager API のユーザとして、XML スキーマについての理解も必要です。XML スキーマの詳細については、http://www.w3.org/TR/xmlschema-0/ を参照してください。

Cisco Unified Communications Manager とそのアプリケーションについての理解も必要です。Cisco Unified Communications Manager のドキュメントやその他の関連技術については、「関連資料」（P.xli）を参照してください。

マニュアルの構成

次の表では、このマニュアルの章の概要を示しています。

章	説明
第 1 章「概要」	この章では、Cisco IP Telephony ソリューションの JTAPI アプリケーションを作成する前に理解しておく必要のある主な概念を説明します。
第 2 章「新機能と変更情報」	この章では、Cisco Unified JTAPI でサポートされている Cisco Unified Communications Manager リリースの新機能および変更された機能について説明します。
第 3 章「Cisco Unified JTAPI でサポートされる機能」	この章では、Cisco Unified JTAPI でサポートされるその他の機能について説明します。
第 4 章「Cisco Unified JTAPI のインストール」	この章では、Cisco Unified JTAPI のインストール プロセスを説明します。
第 5 章「Cisco Unified JTAPI パッケージのすべての階層」	この章では、Cisco Unified JTAPI パッケージのすべての階層を説明します。
第 6 章「Cisco Unified JTAPI 拡張」	この章では、Cisco Unified Communications Manager の実装で使用可能な拡張（インターフェイスとクラス）について説明します。
第 7 章「Cisco Unified JTAPI のアラームとサービス」	この章では、Cisco Unified Communications Manager の実装で使用可能なアラームとサービスについて説明します。
第 8 章「Cisco Unified JTAPI の例」	この章では、makecall（JTAPI インストールをテストするのに使用される Cisco Unified JTAPI プログラム）のソース コードの例を記載しています。
付録 A「メッセージ シーケンスの図」	この付録には、メッセージ フローの図が掲載されています。

章	説明
付録 B「Cisco Unified JTAPI クラスおよびインターフェイス」	この付録には、Cisco Unified JTAPI で利用可能なすべてのクラスおよびインターフェイスの一覧があります。
付録 C「Cisco Unified JTAPI のトラブルシューティング」	この付録には、CTI エラー コード、CiscoEvent ID、およびその他のトラブルシューティング情報が記載されています。
付録 D「リリースごとの Cisco Unified JTAPI オペレーション」	この付録には、Cisco Unified Communications Manager リリースにおけるサポート対象、サポート対象外、変更、検討中または調査中の一覧が記載されています。
付録 E「CTI でサポートされるデバイス」	この付録には、CTI でサポートされるデバイスの一覧が記載されています。
付録 F「定数フィールド値」	この付録では、static final フィールドとその値を示します。
付録 G「非推奨の API」	この付録では、非推奨の API、フィールド、およびメソッドを示します。

関連資料

このセクションでは、Cisco Unified Communications Manager、Cisco Unified IP Phone、およびアプリケーションの開発に必要なテクノロジーに関する情報を提供するドキュメントと URL の一覧を示しています。

- Cisco Unified Communications Manager リリース 7.1(2) : Cisco Unified Communications Manager のインストールと設定に関するドキュメントのセット。Cisco Unified Communications Manager 7.1(2) のインストールおよび設定に関するドキュメントについては、『*Cisco Unified Communications Manager Documentation Guide for Release 7.1(2)*』を参照してください。次のようなドキュメントの情報が記載されています。
  - *Cisco Unified Communications Manager Administration Guide, Release 7.1(2)*
  - *Cisco Unified Communications Manager System Guide, Release 7.1(2)*
  - *Cisco Unified Communications Manager Features and Services Guide, Release 7.1(2)*
- *Cisco Unified IP Phones and Services* : Cisco Unified IP Phone のインストールと設定に関するドキュメントのセット。
- *Cisco Distributed Director* : Cisco DistributedDirector のインストールと設定に関するドキュメントのセット。

関連情報

Sun Microsystems JTAPI 仕様ファイルの最新バージョンすべてを入手するには、直接 http://java.sun.com/products/jtapi を参照してください。

開発者のサポート

Developer Support Program は、Cisco Systems インターフェイスを公式にサポートするもので、Cisco Service Provider のソリューションである Ecosystem および Cisco AVVID Partner プログラムの開発者、カスタマー、およびパートナーが相互にソリューションを提供できるようにします。

Developer Support のエンジニアは、製品技術エンジニアリング チームの拡張メンバーです。このエンジニアは、専門的なサポートを提供するために必要なリソースに対して、直接タイムリーにアクセスします。

このプログラムの詳細については、Developer Support Program の Web サイト（http://developer.cisco.com）を参照してください。

Cisco Unified Communications Manager Release 5.0(1) Extension Mobility API Developer Guide を使用して開発を行う場合は、Cisco Developer Support Program に参加されることをお勧めします。このプログラムでは、開発プロジェクトでシスコのインターフェイスをご利用の際に、一貫したレベルのサポートを提供します。

（注） Cisco Technical Assistance Center（TAC）のサポートには、エクステンション モビリティ API の開発者向けサポートは含まれていません。また、サポートの範囲は、Cisco AVVID のインストールと構成、ならびにシスコ製アプリケーションに限定されています。Developer Support Program の詳細については、developer-support@cisco.com までお問い合わせください。

表記法

このマニュアルでは、次の表記法を使用しています。

表記法	説明
太字	コマンドおよびキーワードは**太字**で示しています。
イタリック体	ユーザが値を指定する引数は、*イタリック体*で示しています。
[ ]	角カッコの中の要素は、省略可能です。
{ x | y | z }	必ずどれか 1 つを選択しなければならない必須キーワードは、波カッコで囲み、縦棒で区切って示しています。
[x | y | z]	どれか 1 つを選択できる省略可能なキーワードは、角カッコで囲み、縦棒で区切って示しています。
ストリング	引用符を付けない一組の文字。ストリングの前後には引用符を使用しません。引用符を使用すると、その引用符も含めてストリングとみなされます。
`screen` フォント	システムが表示する端末セッションおよび情報は、`screen` フォントで示しています。
`太字の screen` フォント	ユーザが入力しなければならない情報は、**`太字の screen`** フォントで示しています。
イタリック体の screen フォント	ユーザが値を指定する引数は、*イタリック体の screen* フォントで示しています。
→	この矢印は、例の中の重要な行やテキストを強調するためのものです。
^	^ 記号は、Ctrl キーを表します。たとえば、画面に表示される ^D というキーの組み合わせは、Ctrl キーを押しながら D キーを押すことを意味します。
< >	パスワードのように出力されない文字は、かぎカッコ（< >）で囲んで示しています。

（注）は、次のように表しています。

（注） 「*注釈*」です。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。

ワンポイント アドバイスは、次のように表しています。

ワンポイントアドバイス *時間を節約する方法*です。ここに紹介している方法で作業を行うと、時間を短縮できます。

ヒントは、次のように表しています。

ヒント *便利なヒント*です。

マニュアルの入手方法、テクニカル サポート、およびセキュリティ ガイドライン

マニュアルの入手方法、テクニカル サポート、マニュアルに関するフィードバックの提供、セキュリティ ガイドライン、および推奨エイリアスや一般的なシスコのマニュアルについては、次の URL で、毎月更新される『What's New in Cisco Product Documentation』を参照してください。シスコの新規および改訂版の技術マニュアルの一覧も示されています。

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

『What's New in Cisco Product Documentation』は RSS フィードとして購読できます。また、リーダー アプリケーションを使用してコンテンツがデスクトップに直接配信されるように設定することもできます。RSS フィードは無料のサービスです。シスコは現在、RSS バージョン 2.0 をサポートしています。

シスコ製品のセキュリティの概要

本製品には暗号化機能が備わっており、輸入、輸出、配布および使用に適用される米国および他の国での法律を順守するものとします。シスコの暗号化製品を譲渡された第三者は、その暗号化技術の輸入、輸出、配布、および使用を許可されたわけではありません。輸入業者、輸出業者、販売業者、およびユーザは、米国および 他の国での法律を順守する責任があります。本製品を使用するにあたっては、関係法令の順守に同意する必要があります。米国および 他の国の法律を順守できない場合は、本製品を至急送り返してください。

Cisco の暗号化製品を管理する米国の法規の概要は http://www.cisco.com/wwl/export/crypto/tool/stqrg.html で参照できます。

さらに詳しい情報が必要な場合は、export@cisco.com 宛てに電子メールでお問い合わせください。

OpenSSL 通告

次のリンクでは、OpenSSL 通告について説明しています。

http://www.cisco.com/en/US/products/hw/phones/ps379/products_licensing_information_listing.html

CHAPTER 1

概要

この章では、Cisco Unified Communications ソリューションの Java Telephony Application Programming Interface（JTAPI）アプリケーションを作成する前に理解しておく必要のある主な概念について説明します。次のような構成になっています。

Cisco Unified Communications Manager の機能については、第 3 章「Cisco Unified JTAPI でサポートされる機能」を参照してください。詳細についてと、CTI のデバイスおよびサポートされる機能については、付録 E「CTI でサポートされるデバイス」と付録 D「リリースごとの Cisco Unified JTAPI オペレーション」を参照してください。

JTAPI の概要

Cisco Unified JTAPI は、Java ベースのコンピュータ テレフォニー アプリケーションとともに使用する目的で、Sun Microsystems によって開発された標準のプログラミング インターフェイスとして機能します。Cisco JTAPI では、Sun JTAPI 1.2 仕様が Cisco の追加の機能拡張として実装されています。Cisco JTAPI を使用して、次のアプリケーションを開発できます。

- Cisco Unified Communications Manager フォンの監視と制御。
- Computer–Telephony Integration（CTI）ポートとルート ポイント（仮想デバイス）を使用したコールのルーティング。

サポートされる基本的なテレフォニー API は、会議、転送、接続、応答、リダイレクト API で構成されます。

javax.telephony.* 階層にある JTAPI インターフェイスのパッケージは、テレフォニー リソースを Java アプリケーションで操作するためのプログラミング モデルを定義しています。インターフェイスの詳細については、付録 B「Cisco Unified JTAPI クラスおよびインターフェイス」を参照してください。

この章では、次のトピックについて取り上げます。

Cisco Unified JTAPI と Contact Center

Cisco Unified JTAPI は、Contact Center で使用し、正しい時間に正しい場所へコールを送信するためにデバイス ステータスの監視とルーティング指示の発行を行い、分析用にコールの統計を取得すると同時に録音指示の停止や開始を指示し、さらに、CRM アプリケーション、自動化スクリプト、およびリモート コール制御内で、コールを画面にポップアップさせます。

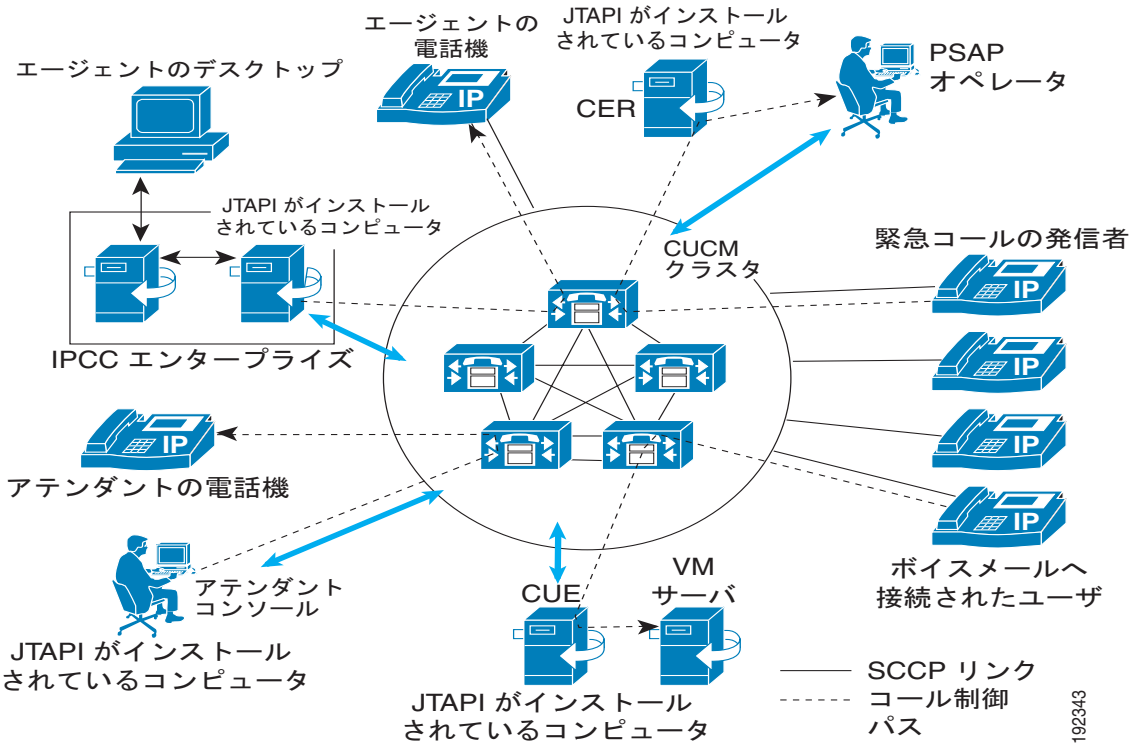
Cisco Unified JTAPI とエンタープライズ

Cisco Unified JTAPI はエンタープライズ環境で使用され、ユーザの応答可能、場所、プリファレンスを結合し、プレゼンス ベースのルーティングを行うための固有に調整された環境を実現します。たとえば、金融環境では、ブローカーや分析者が国際金融市場の急激な変化に対応できるように、市場データ、ビジネス ロジック、コール制御がブラウザ ベースのアプリケーションに組み込まれています。

健康管理環境では、コール制御、医者と患者の検索、救急隊の呼び出しが、ブラウザ ベースのコンソールに組み込まれています。さらに、サービス業環境では、発信者のデータが POS システムにリンクされ、部屋の予約またはレストランの予約、タクシーの手配、およびモーニング コールのスケジューリングなどの自動化が行われます。

図 1-1 は、エンタープライズ用に設定された典型的な Cisco Unified Communications Manager と Cisco Unified JTAPI を示しています。

図 1-1　Cisco Unified Communications Manager と Cisco Unified JTAPI



Cisco Unified JTAPI アプリケーション

Cisco Unified JTAPI アプリケーションのフローは次のようになります。

- JTAPIPeerFactory から JTAPIPeer オブジェクト インスタンスを取得する。
- JTAPIPeer で getProvider() API を使用し、プロバイダーを取得する。
- プロバイダーから、アプリケーションが使用する端末とアドレスを取得する。
- 関連するオブジェクトの機能を判別する。
- アプリケーションが監視および制御する必要がある、オブジェクトのオブザーバを追加する。
- アプリケーション フローを開始する（例：コールの開始）。

次の例に、基本的な JTAPI アプリケーションを示します。

```
    public void getProvider () {
            try {
                        JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
                        System.out.println ("Got peer "+peer);
                        Provider provider =
peer.getProvider("cti-server;login=username;passwd=pass");
                        System.out.println ("Got provider "+provider);
                        MyProviderObserver providerObserver = new MyProviderObserver ();
                        provider.addObserver(providerObserver);
                        while (outOfService ) {
                                Thread.sleep(500);
                        }
                        System.out.println ("Provider is now in service");
                        Address[] addresses = provider.getAddresses();
                        System.out.println ("Found "+ addresses.length +" addresses");
                        for(int i=0; i< addresses.length; i++){
                                System.out.println(addresses[i]);
                        }
                        provider.shutdown();
            } catch (Exception e){
            }

    }
```

Jtprefs アプリケーション

Cisco Unified JTAPI の設定に必要なパラメータは、jtapi.ini ファイルにあります。Cisco Unified JTAPI は、このファイルを Java クラスパス内で検出します。パラメータは、Cisco Unified JTAPI でインストールされる Jtprefs アプリケーションを使用して変更できます。Jtprefs アプリケーションではこのアプリケーションが必要とするパラメータだけが設定されます。jtapi.ini に依存せずに、1 箇所からアプリケーションを管理できるため、大変有用です。

jtapi.ini ファイルにはデフォルト値が含まれていますが、クライアント アプリケーションでは jtapi.ini ファイルを特に修正することなくさまざまな値を修正できます。ただし、クライアント アプリケーションのさまざまなインスタンスでは、これらのパラメータに対して個別の設定を指定できます。CiscoJtapiProperties インターフェイスは、com.cisco.jtapi.extensions パッケージで定義されています。

アプリケーションでは、CiscoJtapiPeer から CiscoJtapiProperties オブジェクトを取得し、アクセス用メソッドと変更用メソッドを使用してパラメータに変更を加えます。これらのプロパティは、CiscoJtapiPeer から派生したすべてのプロバイダーに対して、CiscoJtapiPeer で最初の getProvider () が呼び出される前に、設定および適用する必要があります。

jtprefs.ini を起動できない非 GUI ベースのプラットフォームで実行されるアプリケーションでは、jtapi.ini ファイルを作成して jtapi.jar とともに配置できます。

詳細については、次のトピックを参照してください。

- 「jtapi.ini ファイルのフィールド」（P.4-21）
- 「デフォルト値を使用する場合の jtapi.ini ファイルの例」（P.4-27）

Cisco Unified JTAPI の概念

ここでは、次の概念について説明します。

- 「CiscoObjectContainer インターフェイス」（P.1-4）
- 「JtapiPeer と Provider」（P.1-4）
- 「Address と Terminal の関係」（P.1-6）
- 「Connection」（P.1-7）
- 「Terminal Connection」（P.1-8）
- 「端末とアドレスの制限」（P.1-8）
- 「CiscoConnectionID」（P.1-11）

CiscoObjectContainer インターフェイス

CiscoObjectContainer インターフェイスを使用すると、このインターフェイスの実装されたオブジェクトに、アプリケーションによって定義されたオブジェクトを関連付けることができます。Cisco Unified JTAPI では、次のインターフェイス上で CiscoObjectContainer インターフェイスが拡張されます。

- CiscoJTAPIPeer
- CiscoProvider
- CiscoCall
- CiscoAddress
- CiscoTerminal
- CiscoConnection
- CiscoTerminalConnection
- CiscoConnectionID
- CiscoCallID

JtapiPeer と Provider

JtapiPeer オブジェクトの実装によって作成される Provider オブジェクトは、アプリケーションと JTAPI 実装間の主な接点として機能します。Provider オブジェクトは、アプリケーションで常に制御可能な Address、Terminal、Call などのコール モデル オブジェクト全体の集合を格納しています。

JTAPI Preferences（JTPREFS）アプリケーションでは、サーバ名を返す JtapiPeer.getServices() を管理します。

Provider は 2 つの基本プロセス（初期化とシャットダウン）を伴います。

アプリケーションで CiscoProvider を取得するときには、JtapiPeer.getProvider() メソッドを使用して必ず次の情報を渡します。

- Cisco Unified Communications Manager サーバのホスト名または IP アドレス。
- ディレクトリで管理されるユーザのログイン。
- 指定したユーザのパスワード。
- （オプション）アプリケーション情報（このパラメータには任意の長さの文字列を指定できます）。

  appinfo がアラームにログされた場合に管理者がアラームの原因となったアプリケーションを認識できるように、アプリケーションには十分な情報を記述する必要があります。アプリケーションには、アプリケーションが常駐するホスト名または IP アドレス、およびアプリケーションが起動した時間を含めないでください。また、「=」または「;」の記号は、getProvider() 文字列の区切りに使用されるため、appinfo 文字列に入れることはできません。appinfo を指定しない場合、代わりに一般的な擬似固有名（JTAPI[XXXX]@hostname）を使用できます（XXXX は 4 桁の乱数を表します）。

パラメータは、次のように文字列として連結されたキー値のペアで渡します。

```
JtapiPeer.getProvider("CTIManagerHostname;login=user;passwd=userpassword;appinfo=Cisco Softphone")
```

初期化

JtapiPeer.getProvider() メソッドでは、TCP リンク、Cisco Unified Communications Manager との初期ハンドシェイク、およびデバイス リストの列挙が完了すると同時に Provider オブジェクトが返されます。このとき、プロバイダーは OUT_OF_SERVICE 状態にあります。Cisco Unified JTAPI アプリケーションは、制御対象のデバイス リストが有効になる前に、プロバイダーが IN_SERVICE 状態に移行するのを待つ必要があります。ProvInServiceEv イベントは、ProviderObserver インターフェイスの実装されたオブジェクトに通知されます。

（注） CiscoProviderObserver の実装だけでは不十分であり、provider.addObserver() を使用してプロバイダーにオブザーバを追加する必要があります。アプリケーションは、Provider がイン サービス状態にあることを示す通知を待つ必要があります。

JTAPI における QoS ベースライン化作業の一環として、ProviderOpenCompletedEv は「DSCP value for Applications」を JTAPI に提供します。JTAPI は CTI との接続に対してこの DSCP 値を設定し、Provider オブジェクトが存在する限り、CTI へのすべての JTAPI メッセージがこの DSCP 値を保持します。

シャットダウン

アプリケーションを使用して provider.shutdown() を呼び出すと、JTAPI では Cisco Unified Communications Manager との通信が恒久的に失われ、ProvShutdownEv イベントがアプリケーションに通知されます。Provider が再度起動されることはないと想定し、アプリケーションによってシャットダウンを完全に処理する必要があります。

Provider.getTerminals()

このメソッドは、ディレクトリ上にあるユーザ制御リストで管理されるデバイスに対して作成される端末の配列を返します。ユーザ制御リストの管理については、*Cisco Unified Communications Manager Administration Guide* を参照してください。

Provider.getAddresses()

このメソッドは、ディレクトリ上にあるユーザ制御リストで管理されるデバイスに対して割り当てられる回線から作成されるアドレスの配列を返します。

ディレクトリ上のユーザ制御リストの変更

JTAPI アプリケーションの起動後にユーザ制御リストにデバイスが追加された場合、CiscoTermCreatedEv と各 CiscoAddrCreatedEv が生成され、CiscoProviderObserver の実装されたオブザーバに送信されます。また、アプリケーション側から、制御対象デバイスの現在の登録状態を監視し、これらのデバイスのアベイラビリティを動的に追跡することができます。イン サービスの Address または Terminal に対するイベントは、CiscoAddressObserver と CiscoTerminalObserver を実装しているオブザーバに通知されます。

（注） オブザーバの実装だけでは不十分であり、アドレスの場合は address.addObserver() メソッドを使用して、端末の場合は terminal.addObserver() メソッドを使用してオブザーバをそれぞれ追加する必要があります。

（注） call.connect() メソッドを呼び出す前に、発呼側のアドレスまたは端末に CallObserver を追加します。追加しないと、このメソッドによって例外が返されます。

Address と Terminal の関係

Cisco Unified Communications システムのアーキテクチャには 3 種類の基本エンドポイントがあります。

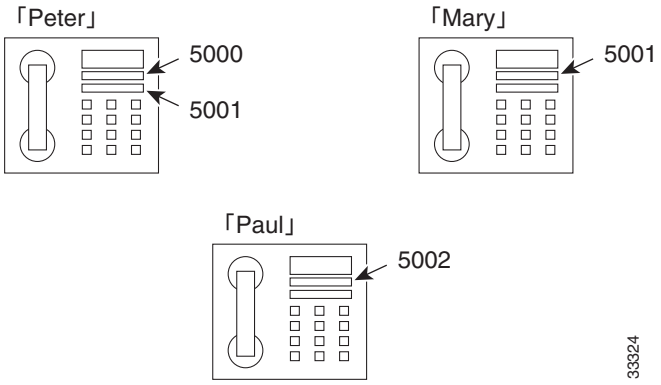
- 電話機
- 仮想デバイス（メディア終端点とルート ポイント）
- ゲートウェイ

これらのエンドポイントのうち、電話機とメディア終端点だけが Cisco Unified JTAPI を実装することにより使用されます。

Cisco Unified Communications Manager を使用するとユーザは、1 つ以上の回線やダイヤル可能な番号を電話機に設定して、その回線や番号を複数の電話機間で同時に共用したり、複数の回線を一度に 1 台の電話機だけで排他的に利用するように設定することができます。電話機には、回線あたり同時に複数のコールを終端させる機能（Maximum Number of Calls の設定に依存）があり、1 つのコール以外は保留になります。

これは、家庭用電話機の「コール ウェイティング」機能の動作に似ています。図 1-2 は、Peter と Mary が 1 本の電話回線 5001 を共有する設定と、Paul が専用の電話回線 5002 を使用する設定を示したものです。

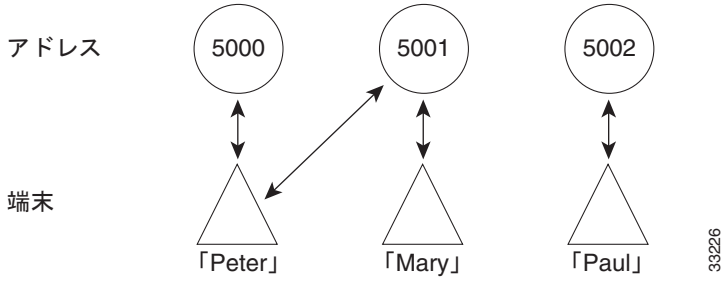
図 1-2　電話機の図



Cisco Unified Communications Manager では、あらゆる種類のエンドポイントが固有の名前によって識別されます。電話機の Media Access Control（MAC; メディア アクセス制御）アドレス（「SEP0010EB1014」など）で識別できますが、システム管理者は、区別可能であればどのような名前でもメディア終端点に割り当てることができます。

Cisco Unified JTAPI 実装では、プロバイダーによって制御される各エンドポイントごとに、管理者が割り当てた名前を使用して、対応する端末オブジェクトが構築されます。端末オブジェクトにもまた 1 つ以上のアドレス オブジェクトがあり、各アドレス オブジェクトはエンドポイント上の回線に対応します。図 1-3 の「Address と Terminal の関係」に Address と Terminal の関係を図示します。

図 1-3　Address と Terminal の関係



2 つ以上のエンドポイントで 1 本の回線（電話番号）を共用する場合は、対応するアドレス オブジェクトは 2 つ以上の端末オブジェクトに関連付けられます。

観察対象外の Address と Terminal

Cisco Unified JTAPI では、プロバイダーの端末とアドレスに CallObserver が付けられている場合にだけ、コールが認識されます。これは、Provider.getCalls() または Address.getConnections() などのメソッドでは、アドレスにコールがあってもこのアドレスに CallObserver が付けられていない場合は null が返されることを意味します。また、Call.connect() メソッドを使用して発呼するアドレスや端末にも、CallObserver を追加する必要があります。

Connection

Connection では、コールとアドレスへの恒久的な参照が保持されます。このため、コール イベントから取得した Connection の参照は、常に Connection のコール（getCall()）とアドレス（getAddress()）の取得に使用できます。

Terminal Connection

Terminal Connection では、常に端末と Connection への参照が保持されます。このため、コール イベントから取得した Terminal Connection の参照は、常に接続端末（getTerminal()）と Connection（getConnection()）の取得に使用できます。

端末とアドレスの制限

この端末とアドレスの制限では、管理者が Cisco Unified Communications Manager Administration の制限リストに特定の端末とアドレスのセットを追加した場合、それらの端末とアドレスをアプリケーションで制御または監視することが禁止されます。

管理者はデバイス上の特定の回線（特定の端末上のアドレス）を制限リストに追加できます。Cisco Unified Communications Manager Administration の制限リストに端末を追加した場合は、JTAPI でもその端末のすべてのアドレスが制限付きとしてマークされます。この設定が完了した後にアプリケーションを起動した場合は、CiscoTerminal.isRestricted() インターフェイスと CiscoAddress.isRestricted(Terminal) インターフェイスを確認することで、特定の端末またはアドレスが制限されているかどうかを認識することができます。共用回線の場合は、CiscoAddress.getRestrictedAddrTerminals() インターフェイスを照会することで、特定のアドレスがいずれかの端末で制限されているかどうかを確認できます。

アプリケーションの起動後に回線（端末上のアドレス）が制限リストに追加された場合は、CiscoAddrRestrictedEv がアプリケーションに通知されます。そのアドレスにオブザーバがある場合は CiscoAddrOutOfService がアプリケーションに通知されます。制限リストから回線が削除されると、CiscoAddrActivatedEv がアプリケーションに通知されます。そのアドレスにオブザーバがある場合は CiscoAddrInServiceEv がアプリケーションに送信されます。アプリケーションが制限リストに含まれるアドレスにオブザーバを追加しようとすると、PlatformException がスローされます。アドレスが制限される前に追加されたオブザーバはそのまま残りますが、そのアドレスが制限リストから削除されない限り、これらのオブザーバでイベントを取得できません。アプリケーションからアドレスのオブザーバを削除することもできます。

アプリケーションの起動後にデバイス（端末）が制限リストに追加された場合は、CiscoTermRestrictedEv がアプリケーションに送信されます。その端末にオブザーバがある場合は CiscoTermOutOfService がアプリケーションに送信されます。端末が制限リストに追加されると、JTAPI でもその端末に属するすべてのアドレスが制限され、CiscoAddrRestrictedEv がアプリケーションに通知されます。制限リストから端末が削除されると、CiscoTermActivatedEv と、対応するアドレスの CiscoAddrActivatedEv がアプリケーションに通知されます。アプリケーションが制限リストに含まれる端末にオブザーバを追加しようとすると、PlatformException がスローされます。端末が制限される前に追加されたオブザーバはそのまま残りますが、その端末が制限リストから削除されない限り、これらのオブザーバでイベントを取得できません。

アプリケーションの起動後に共用回線が制限リストに追加された場合は、CiscoAddrRestrictedOnTerminalEv がアプリケーションに通知されます。そのアドレスにアドレス オブザーバがある場合は、その端末の CiscoAddrOutOfServiceEv がアプリケーションに通知されます。すべての共用回線が制限リストに追加された場合は、最後の 1 つが追加された時点で、CiscoAddrRestrictedEv がアプリケーションに通知されます。アプリケーションの起動後に制限リストから共用回線が削除された場合は、CiscoAddrActivatedOnTerminalEv がアプリケーションに通知されます。そのアドレスにオブザーバがある場合は、その端末の CiscoAddrInServiceEv がアプリケーションに通知されます。制御リストに含まれるすべての共用回線が制限リストから削除された場合は、最後の 1 つが削除された時点で CiscoAddrActivatedEv がアプリケーションに通知され、端末上のすべてのアドレスが InService イベントを受信します。

制御リストに含まれるすべての共用回線が制限付きとしてマークされている場合、アプリケーションがオブザーバを追加しようとすると、PlatformException がスローされます。一部の共用回線のみが制限リストに追加されている場合、アプリケーションがそのアドレスにオブザーバを追加すると、制限されていない回線のみがイン サービスになります。

アドレスまたは端末が制限リストに追加されてリセットされたときにアクティブ コールが存在していた場合は、接続と TerminalConnections が接続解除として通知されます。

アドレスと端末が制限リストに 1 つも追加されていない場合、この機能は JTAPI の以前のバージョンと下位互換性をそのまま維持し、新しいイベントはアプリケーションに配信されません。

次のセクションでは、アドレスおよび端末の制限に関するインターフェイスの変更点について説明します。

CiscoTerminal

boolean	**isRestricted()** 端末が制限されているかどうかを示します。端末が制限されている場合、その端末に関連付けられたすべてのアドレスも制限されます。端末が制限されている場合は true を返します。端末が制限されていない場合は false を返します。

CiscoAddress

javax.telephony.Terminal[]	**getRestrictedAddrTerminals()** このアドレスが制限されている端末の配列を返します。制限されている端末がない場合、このメソッドは null を返します。 共用回線の場合、端末上の少数の回線が制限されている可能性があります。このメソッドは、このアドレスが制限されているすべての端末を返します。アプリケーションは、制限された回線のコール イベントを認識できません。制限された回線が他の制御デバイスとのコールに関わっている場合、制限された回線用の外部接続が作成されます。
boolean	**isRestricted**(javax.telephony.Terminal terminal) この端末上のいずれかのアドレスが制限されている場合、true を返します。この端末上のアドレスが制限されていない場合、false を返します。

```
public interface CiscoRestrictedEv extends CiscoProvEv {
    public static final int ID = com.cisco.jtapi.CiscoEventID.CiscoRestrictedEv;

    /**
     * The following define the cause codes for restricted events
     */

    public final static int CAUSE_USER_RESTRICTED = 1;

    public final static int CAUSE_UNSUPPORTED_PROTOCOL = 2;

    }
```

これは制限されたイベントの基底クラスを表し、すべての制限されたイベントの原因コードを定義します。CAUSE_USER_RESTRICTED は、端末またはアドレスが制限とマークされていることを示します。CAUSE_UNSUPPORTED_PROTOCOL は、制御リスト内のデバイスが Cisco Unified JTAPI でサポートされていないプロトコルを使用していることを示します。SIP を実行している既存の Cisco Unified IP 7960 フォンおよび 7940 フォンはこれに該当します。

CiscoAddrRestrictedEv

public interface **CiscoAddrRestrictedEv** extends CiscoRestrictedEv。アプリケーションは、回線または関連するデバイスが Cisco Unified Communications Manager Administration から制限されたときに、このイベントを通知します。制限された回線では、アドレスがアウト オブ サービスになり、再び有効になるまでイン サービスに戻りません。アドレスが制限されている場合は、addCallObserver および addObserver によって例外がスローされます。共用回線では、いくつかの回線だけが制限されて残りは制限されなかった場合、例外はスローされませんが、制限された共用回線はイベントを受け取りません。すべての共用回線が制限された場合は、オブザーバを追加すると例外がスローされます。オブザーバを追加した後にアドレスが制限された場合、アプリケーションは CiscoAddrOutOfServiceEv を認識し、アドレスが有効にされるとイン サービスになります。

CiscoAddrActivatedEv

public interface **CiscoAddrActivatedEv** extends CiscoProvEv。アプリケーションは、回線または関連するデバイスが制御リストに含まれており、Cisco Unified Communications Manager Administration の制限リストから削除されたときに、このイベントを識別します。該当アドレスにオブザーバが存在する場合、アプリケーションは CiscoAddrInServiceEv を識別します。オブザーバが存在しない場合、アプリケーションはオブザーバの追加を試みることが可能で、アドレスはイン サービス状態になります。

CiscoAddrRestrictedOnTerminalEv

public interface **CiscoAddrRestrictedOnTerminalEv** extends CiscoRestrictedEv。ユーザが制御リストに共有アドレスを持っており、いずれかの回線を制限リストに追加した場合、このイベントが送信されます。getTerminal() インターフェイスは、アドレスが制限される端末を返します。getAddress() インターフェイスは、制限されるアドレスを返します。

javax.telephony.Address	**`getAddress()`**
javax.telephony.Terminal	**`getTerminal()`**

CiscoAddrActivatedOnTerminal

public interface **CiscoAddrActivatedOnTerminalEv** extends CiscoProvEv。共用回線または共用回線を持つデバイスを制限リストから削除すると、このイベントが送信されます。getTerminal() インターフェイスは、アドレスに追加される端末を返します。getAddress() インターフェイスは、新しい端末が追加されるアドレスを返します。

javax.telephony.Address	**`getAddress()`**
javax.telephony.Terminal	**`getTerminal()`**

CiscoTermRestrictedEv

public interface **CiscoTermRestrictedEv** extends CiscoRestrictedEv。アプリケーションは、アプリケーションの起動後にデバイスが Cisco Unified Communications Manager Administration から制限リストに追加されたときに、このイベントを認識します。アプリケーションは、制限された端末やその端末のアドレスに関するイベントを識別できません。InService 状態にある端末が制限された場合、アプリケーションはこのイベントを受信し、端末およびそれに対応するアドレスはアウト オブ サービス状態になります。

CiscoTermActivatedEv

public interface **CiscoTermActivatedEv** extends CiscoRestrictedEv。

javax.telephony.Terminal　**getTerminal()**

有効化されて制限リストから削除された端末を返します。

CiscoOutOfServiceEv

static int　**CAUSE_DEVICE_RESTRICTED**

デバイスが制限されたためにイベントが送られたかどうかを示します。

static int　**CAUSE_LINE_RESTRICTED**

回線が制限されたためにイベントが送られたかどうかを示します。

CiscoCallEv

static int　**CAUSE_DEVICE_RESTRICTED**

デバイスが制限されたためにイベントが送られたかどうかを示します。

static int　**CAUSE_LINE_RESTRICTED**

回線が制限されたためにイベントが送られたかどうかを示します。

CiscoConnectionID

CiscoConnectionID オブジェクトは、Cisco Unified JTAPI の各接続に関連付けられた固有のオブジェクトを表します。アプリケーションでは、オブジェクト自体またはオブジェクトの整数表現を使用できます。

コールバックのスレッド化

Cisco Unified JTAPI 実装の設計では、アプリケーションによって Call.connect() と TerminalConnection.answer() などのブロッキング用 JTAPI メソッドを、そのオブザーバのコールバック内から呼び出すことが可能です。これは、オブザーバのコールバック内から JTAPI メソッドを使用しないように警告する JTAPI 1.2 仕様の制限に、アプリケーションが制約されないことを意味します。

CiscoSynchronousObserver インターフェイス

Cisco Unified JTAPI 実装では、アプリケーションによって Call.connect() と TerminalConnection.answer() などのブロッキング用 JTAPI メソッドを、オブザーバのコールバック内から呼び出すことが可能です。これは、オブザーバのコールバック内部から JTAPI メソッドを使用しないように警告する JTAPI 1.2 仕様の制限に、アプリケーションが制約されないことを意味します。アプリケーションでは、そのオブザーバ オブジェクト上に CiscoSynchronousObserver インターフェイスを実装して、Cisco Unified JTAPI 実装のキューイング ロジックを選択的に無効にできます。

多くのアプリケーションは、この非同期動作の悪影響を受けることはありません。オブザーバ コールバック中にコヒーレント コール モデルの機能を使用するアプリケーションでは、Cisco Unified JTAPI 実装のキューイング ロジックを選択的に無効にできます。対象のオブザーバ オブジェクト上に CiscoSynchronousObserver インターフェイスを実装することで、アプリケーションからそのオブザーバに通知同期イベントが宣言されます。同期オブザーバに通知されるイベントは、オブザーバ コールバック内から照会されるコール モデル オブジェクトの状態と一致します。

（注）　CiscoSynchronousObserver インターフェイスの実装されたオブジェクトでは、そのイベント コールバック内部からブロッキング用 JTAPI メソッドを呼び出さないでください。これを行った場合の影響は予測不能で、JTAPI 実装がデッドロックに陥る可能性があります。ただし、これらのオブジェクトでは、インスタンス Call.getConnections() または Connection.getState() のすべての JTAPI オブジェクトのアクセス用メソッドは安全に使用できます。

動的オブジェクトを照会する

コール オブジェクトのような動的オブジェクトの照会には注意が必要です。イベントを取得する時間までに、オブジェクト（コールなど）は示された状態とは別の状態で存在している場合があります。たとえば、CiscoTransferStartEV を取得する時間までに、転送コールによってその内部接続がすべて削除されている場合があります。

callChangeEvent()

callChangedEvent() メソッドが呼び出されときには、イベントに格納された参照の妥当性は引き続き保証されます。たとえば、イベントに getConnection() メソッドがある場合、アプリケーションでこのメソッドを呼び出して有効な接続の参照を取得できます。同様に、getCallingAddress() メソッドでは有効な Address オブジェクトを返すことが保証されます。

CiscoConsultCall

CiscoConsultCall インターフェイスの場合、コンサルティング端末接続への参照が恒久的に保持されます。たとえば、CiscoConsultCallActive イベントの処理時では、getConsultingTerminalConnection() は有効な端末接続の参照を返すことを保証します。また、端末接続は、コンサルティング接続とそのコンサルティング コールへのアクセスを保証します。

CiscoTransferStartEv

CiscoTransferStartEv の場合、callChangedEvent() が呼び出されると、イベント内にある転送コール、転送コントローラおよび最後のコールへの参照が有効になります。ただし、getConnections() を呼び出したときには、getConnections() から接続が返される場合と返されない場合があります。

アラーム サービス

Cisco Unified Communications アプリケーションに対する一般的なサービス フレームワークの一部として、サービスへのアラームの送信がサポートされています。com.cisco.services.alarm パッケージでは、アラーム コンポーネントが定義されています。

アラームのインターフェイスとフレームワークでは、Cisco Unified JTAPI アプリケーションのネットワーク上で使用可能なアラーム サービスへ、XML 形式のアラーム通知を TCP 経由で送信することがサポートされます。アラーム パッケージには、次の機能があります。

- アラーム サービスのカタログで解決される、アラームの XML 定義
- 送信側でアラームをバッファする、制限があり上書きされるキュー
- 送信アプリケーション側でブロッキングを回避する、別のスレッド上でのアラーム送信
- アラーム サービスへの TCP ベースの再接続スキーム

Cisco Unified JTAPI アラーム システムの全体的なフレームワークは、既存の JTAPI トレース パッケージに似た点が含まれています。アプリケーションでは、アラーム オブジェクトを作成可能な特定のファシリティ コード用の AlarmManager をインスタンスにする必要があります。実装の一部として、DefaultAlarm および DefaultAlarmWriter 実装クラスが含まれています。

ソフトウェア要件

次の表では、JTAPI アプリケーション、JTPREFS、およびサンプル コードのソフトウェア要件を示しています。

アプリケーション	ソフトウェア要件	例
JTAPI アプリケーション	いずれかの JDK 1.4.2 対応 Java 環境	• Internet Explorer 4.01 以降 • Sun JDK 1.4.2 または 1.5
JTPREFS	いずれかの JDK 1.4.2 対応環境	
コード例	Microsoft Internet Explorer 4.01 以降	

CHAPTER 2

新機能と変更情報

この章では、Cisco Unified Communications Manager リリース 7.1(2) の JTAPI に関する新規情報および改訂情報と、前のリリースでサポートされている機能について説明します。次のような構成になっています。

詳細については、
http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html の Programming Guides Web サイトを参照してください。

Cisco Unified Communications Manager リリース 7.1(2)

このセクションでは、Cisco Unified Communications Manager リリース 7.1(2) の新機能と変更された機能について説明します。

- 任意の通話者のドロップ（Drop Any Party）：電話会議から任意の通話者をドロップする機能。詳細は、「任意の通話者のドロップ（Drop Any Party）」（P.3-1）を参照してください。
- IPv6：IPv6 アドレスがサポートされ、Cisco JTAPI は CTIManager への IPv6 接続をサポートするように拡張されました。詳細は、「IPv6 のサポート」（P.3-2）を参照してください。
- 回線をまたいで直接転送（Direct Transfer Across Lines）：デバイス上に設定された回線をまたいでコールを直接転送できます。詳細は、「回線をまたいで直接転送（Direct Transfer Across Lines）」（P.3-3）を参照してください。
- 回線をまたいで参加（Join Across Lines）または Connected Conference Across Lines：この機能はこのリリースで拡張されました。新しく導入された電話モデルでは、既存の回線をまたいで参加（Join Across Lines）とは異なり、サービス パラメータを使用せず、常に有効となります。詳細は、「回線をまたいで参加（Join Across Lines）または Connected Conference Across Lines」（P.3-8）を参照してください。
- スワップまたはキャンセル、および転送または会議の動作：サポートされている IP フォンでのスワップおよびキャンセルが Cisco Unified JTAPI でサポート可能になりました。詳細は、「スワップまたはキャンセルと転送または会議の動作」（P.3-11）を参照してください。
- 拡張された MWI：アプリケーションがメッセージをカウントし、拡張されたメッセージ受信数をサポートしている電話機で表示できるようになりました。詳細は、「メッセージ受信インジケータの拡張」（P.3-13）を参照してください。

- パーク モニタリングと Assisted DPark のサポート：新しいパークの復帰動作が提供され、アプリケーションは新しい電話機からのパーク要求を呼び出せるようになりました。詳細は、「パーク監視と Assisted DPark のサポート」（P.3-13）を参照してください。
- 論理パーティション設定：管理者が地域を設定し、VoIP の電話機に直接接続されている PSTN ゲートウェイまたは別の地域の VoIP PSTN ゲートウェイをパス スルーするコールを制限できます。詳細は、「論理パーティション化」（P.3-15）を参照してください。
- コンポーネントのアップデータ：この機能は、アプリケーションがアップデータのログの場所を指定できるように拡張された。現在のアップデータのログは、アプリケーションと同じディレクトリに作成されます。詳細は、「Component Updater」（P.3-16）を参照してください。
- Cisco Unified IP Phone 6900 シリーズのサポート：Cisco Unified JTAPI アプリケーションでロール オーバー モードを有効にして端末を制御できます。詳細は、「Cisco Unified IP Phone 6900 シリーズのサポート」（P.3-16）を参照してください。

以前のリリースでサポートされている機能

このセクションでは、7.1(2) よりも前のリリースでサポートされている機能について説明します。次のような

内容で構成されています。

Cisco Unified Communications Manager リリース 7.0(1)

このセクションでは、Cisco Unified Communications Manager のリリース 6.1 からリリース 7.0(1) および Cisco Unified JTAPI の拡張機能の新機能と変更された機能について説明します。このセクションの内容は次のとおりです。

（注）　Cisco Unified Communications Manager リリース 7.0(1) では、次の IPv6 関連のメソッドをサポートしていません。

canSupportIPv6()
setProviderOpenRetryAttempts (int retryAttempts)
getProviderOpenRetryAttempts()
getIPAddressingMode()（*CiscoMediaTerminal* と *CiscoRouteTerminal* インターフェイスで使用できる）
register(java.net.InetAddress address, int port, CiscoMediaCapability [] capabilities, int[] algorithmIDs, java.net.InetAddress address_v6, int activeAddressingMode)
register(CiscoMediaCapability [] capabilities, int[] int registration Type, int[] algorithmIDs, int activeAddressingMode)
getTerminals()（*新しいインターフェイス CiscoProviderTermCapabilityChangedEv* で使用できる）
getAddressingModeForMedia()
getCallingPartyIpAddr_v6()（*CiscoCallCtlConnOfferedEv* と *CiscoRouteEvent* インターフェイスで使用できる）
CTIERR_IPADDRMODEMISMATCH
CTIERR_DYNREG_IPADDRMODE_MISMATCH
hasIPv6CapabilityChanged()
CiscoTerminal.IP_ADDRESSING_MODE_IPv4
CiscoTerminal.IP_ADDRESSING_MODE_IPv6
CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6
CiscoTerminal.IP_ADDRESSING_MODE_Unknown
CiscoTermRegistrationFailedEv.IP_ADDRESSING_MODE_MISMATCH

（注） 回線をまたいで参加（Join Across Lines）、Do Not Disturb-Reject、および発信側の正規化機能を使用する場合、各 JTAPI アプリケーションをこれらの機能と互換性があるバージョンにアップグレードする必要があります。また、リリース 5.1 からアップグレードして回線をまたいで参加（Join Across Lines）を使用する場合、すべてのアプリケーションを CUCM の新しいバージョンと互換性のあるバージョンにアップグレードするまで、会議のチェーニング機能を有効化または使用しないでください。さらに、アプリケーションが会議のチェーニング機能によって影響を受けないことを確認してください。

Cisco Unified Communications Manager リリース 6.1

このセクションでは、Cisco Unified Communications Manager リリース 6.0 からリリース 6.1(1) および Cisco Unified JTAPI の拡張機能の新機能と変更された機能について説明します。このセクションの内容は次のとおりです。

- 「証明書ダウンロードの API 機能拡張」（P.3-22）
- 「回線をまたいで参加（Join Across Lines）」（P.3-23）
- 「エクステンション モビリティのインターコム サポート」（P.3-23）

Cisco Unified Communications Manager リリース 6.0

このセクションでは、Cisco Unified Communications Manager リリース 6.0 と Cisco Unified JTAPI の拡張機能の新機能と変更された機能について説明します。このセクションの内容は次のとおりです。

- 「録音とサイレント モニタリング」（P.3-24）
- 「インターコム」（P.3-26）
- 「アラビア語とヘブライ語の言語サポート」（P.3-28）

Cisco Unified Communications Manager リリース 5.1

このセクションでは、Cisco Unified Communications Manager のリリース 5.0 からリリース 5.1 および Cisco Unified JTAPI の拡張機能の 新機能と変更された機能について説明します。このセクションの内容は次のとおりです。

Cisco Unified Communications Manager リリース 5.0

このセクションでは、Cisco Unified Communications Manager のリリース 4.x からリリース 5.0 および Cisco Unified JTAPI の拡張機能の新機能と変更された機能について説明します。このセクションの内容は次のとおりです。

下位互換性

このリリースの JTAPI は、Cisco Unified Communications Manager リリース 6.0 用に作成されたアプリケーションと下位互換性があります。CiscoJtapiClient のアップグレードは必須ではありません。

Cisco JtapiClient のアップグレードは必須ではありません。アプリケーションを Cisco Unified Communications Manager リリース 6.1、CiscoJTAPIClient にアップグレードする必要があるのは、Cisco Unified Communications Manager リリース 6.1 から導入された新しい機能をすべて使用する場合だけです。

ユーザが Cisco Unified Communications Manager 4.x (Windows) から Cisco Unified Communications Manager 7.0(1) (Linux) にアップグレードした場合は、jtapi.jar をダウンロードするためのプラグイン URL が異なります。Cisco Unified Communications Manager リリース 7.0(1) のプラグイン URL は、リリース 6.x のプラグイン URL と同じです。

アプリケーションは次のルールに従う必要があります。

- Cisco Unified JTAPI イベントの順序は、JTAPI プロトコルの処理に必要かどうかに関係なく、変更される可能性があります。アプリケーションの開発者は、JTAPI プロトコルの処理に必要かどうかに関係のないイベントの順序に従わないでください。アプリケーションは、その順序が（API Name）プロトコルの処理に必要な場合でも、順序の変わってしまったイベントを復元できるようにする必要があります。
- アプリケーションは、推奨されないメソッドの使用は避ける必要があります。
- アプリケーションは、新しい電話機では新しい動作やイベントの順序が返される可能性があることを考慮する必要があります。
  - アプリケーションで処理しないイベントは無視する必要があります。例：アプリケーションが CiscoTransferStart と End イベントの使用を選択した場合、CiscoCallChangedEv は無視される場合があります。
  - アプリケーションには、認識されていない理由に対する、デフォルトの動作を用意しておく必要があります。

- 未定義のエラー コードに対するデフォルトの動作を用意しておく必要があります。
- JTAPI 1.2 仕様に定義されておらず、『*Cisco Unified JTAPI Developers Guide*』にも記載のない動作については、機能とは見なさないでください。

Cisco Unified Communications Manager Administration リリース 4.x (Windows) の場合、URL は http://<サーバ名または IP アドレス>/CCMPluginsServer/jtapi.jar です。

Cisco Unified Communications Manager Administration リリース 6.x と 7.0(1) (Linux) の場合、URL は

http://<サーバ名または IP アドレス>/plugins/jtapi.jar です。

Cisco Unified Communications Manager リリース 4.x と Cisco Unified Communications Manager リリース 7.0(1) 間では、プラグイン URL の下位互換性はありません。

CHAPTER 3

Cisco Unified JTAPI でサポートされる機能

この章では、Cisco Unified JTAPI 仕様でサポートする機能について説明し、次のトピックについて説明します。

任意の通話者のドロップ（Drop Any Party）

この機能は、任意の参加者を電話会議から終了させる機能です。Cisco Unified JTAPI により、アプリケーションが接続のためにアドレスを監視していない場合でも、既存のインターフェイス Connection.disconnect() を使用して、参加者を会議から終了させることができます。これまで、アプリケーションでは、アドレスが監視対象のアドレスである接続のみ切断できました。

機能の動作は、Unified CM サービス パラメータ Advanced Ad Hoc Conference Enabled の設定に基づいて異なります。このサービス パラメータが False に設定されている場合、アプリケーションは会議コントローラのアドレスを監視している場合にのみ、電話会議の監視していないアドレスの接続を終了できます。このパラメータが True に設定されている場合、アプリケーションは無制限で接続を終了できます。

Cisco Unified JTAPI は、CiscoConnection で、接続のための CiscoPartyInfo オブジェクトの配列を取得するインターフェイスを提供します。CiscoPartyInfo は、CiscoConnection で提供されている新しいインターフェイス disconnect() を使用して、参加者を会議から切断させることができます。通常の回線では、その接続に CiscoPartyInfo が 1 つしかありませんが、共用回線には、共用回線の回線ごとに 1 つの CiscoPartyInfo があります。これにより、電話会議に複数の共用回線の参加者がいる場合に、アプリケーションは共用回線の参加者を選択して切断できます。共用回線の参加者の接続は 1 つしかないため、アプリケーションで既存の Connection.disconnect() API を使用した場合、すべての共用回線の参加者が切断されます。

Cisco Unified JTAPI では、CiscoJtapiProperties で、この機能を有効または無効にするインターフェイス setDropAnyPartyEnabled() を提供しています。デフォルトでこれは有効になっています。または、アプリケーションは、jtapi.ini ファイルの JTAPI ini パラメータ dropAnyPartyEnabled=0 を使用して、任意の通話者のドロップ（Drop Any Party）機能を無効にし、dropAnyPartyEnable=1 を使用して、この機能を有効にできます。jtapi.ini ファイルに dropAnyPartyEnable パラメータが存在しない場合、この機能はデフォルトで有効になります。

Cisco Unified JTAPI では、CiscoCall で、コールが電話会議であるかどうかを判断するためのインターフェイス isConferenceCall() も提供しています。この簡単なメソッドはブール値を返します。

インターフェイスの変更

「CiscoCall」（P.6-70）および「CiscoConnection」（P.6-123）を参照してください。

メッセージ シーケンス

「任意の通話者のドロップ（Drop Any Party）の使用例」（P.A-233）を参照してください。

下位互換性

この機能は下位互換性があります。

IPv6 のサポート

この機能は、IPv6 アドレスをサポートし、CiscoUnified JTAPI が CTIManager への IPv6 接続をサポートするように拡張されています。この機能により、Cisco Unified JTAPI アプリケーションは、IPv6 のサポート機能が有効にされ、発信側が IPv6 対応電話を使用している場合に、発信側アドレスとして、IPv6 アドレスを認識することができます。この機能では、次の関数をサポートしています。

- Cisco Unified JTAPI では、CiscoProviderCapabilities インターフェイスで、Cisco Unified Communications Manager の設定で IPv6 がサポートされているかどうかを示す新しい API canSupportIPv6() を公開しています。
- 以前に登録されている設定と現在の設定に不一致がある場合、Cisco Unified JTAPI はメディアまたはルート端末を閉じます。CiscoTermRegistrationFailedEv および新しい原因コード IP_ADDRESSING_MODE_ MISMATCH がこのシナリオに従って送信されます。
- 端末の IP アドレス機能は、CiscoTerminal インターフェイスの getIPAddressingMode() によって API に公開されています。IP アドレス機能は、CiscoTerminal/CiscoMediaTerminal および CiscoRouteTerminal で使用できます。
- IPv6 発信側 IP アドレスは、IPv4 対応デバイスの場合に IPv4 アドレスが提供されるのと同様に、CallCtlConnOfferedEv および RouteEvent の Cisco の拡張機能として InetAddress オブジェクトで提供されます。

監視対象のデバイスが IPv6 デバイスの場合に、CiscoRTPOutputStartedEv および CiscoRTPInputStartedEv の RTP アドレスにも IPv6 アドレスがあります。つまり、CiscoRTPInputProperties の API getLocalAddress() と CiscoRTPOutputProperties の API getRemoteAddress() で、IPv6 形式の IP アドレスを返せるようになりました。API は InetAddress オブジェクトを返し、アプリケーションでは、それが Inet4Address のインスタンスであるか、Inet6Address のインスタンスであるかを確認して、IPv4 形式の IP アドレスか、IPv6 形式の IP アドレスかを判断できます。

アプリケーションでは、IP アドレッシング モードを変更したら、デバイスをリセットする必要があります。そうしないと、予想した結果にならないことがあります。

リリース 7.1 より、Cisco Unified JTAPI では、CiscoTerminal で getIPAddressingMode() API を提供しています。CTI ポートおよびルートポイントの getIPAddressingMode() API もこのリリースからサポートされています。

Cisco Unified JTAPI では CiscoTerminal の同じ API を拡張し、この API は Unified CM の管理ページに、IP フォンの設定済みの IP アドレッシング モードを返します。デバイスの登録後に、ユーザが Unified CM の管理ページから、IP アドレッシング モードを変更した場合、デバイスをリセットする必要があります。更新された値は、IP フォンのリセット後にのみ Cisco Unified JTAPI で参照可能になります。設定済みの IP アドレッシング モードがデュアル スタックである場合、つまり、IPv4 アドレスと IPv6 アドレスがサポートされている場合、電話はこれらのいずれかまたは両方のアドレスで登録できます。これは、ネットワークのタイプと Unified CM の IPv6 のサポートなどの状況によって異なります。IP アドレッシング モードがデュアル スタックである場合、CiscoTerminal の getIPAddressingMode() は CiscoTerminal.IP_ADDRESSING_MODE_IPV4_V6 を返します。

インターフェイスの変更

「CiscoTerminal」（P.6-307）を参照してください。

メッセージ シーケンス

「IPv6 のサポート」（P.A-184）および「IPv6 のサポート」（P.A-296）を参照してください。

下位互換性

この機能は下位互換性があります。

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

回線をまたいで直接転送（Direct Transfer Across Lines）機能により、回線をまたいだ転送をサポートできます。この機能により、電話の [転送（Transfer）] ソフトキーを押すか、Cisco Unified JTAPI で提供されている transfer() API を使用して、同じ端末の異なるアドレスの 2 つのコールを転送できます。回線をまたいで転送を実行すると、アプリケーションは最後のコールとコンサルト コールで共通のコントローラを認識しないため、JTAPI アプリケーションの動作が変わります。API 自体に変更はなく、会議に参加するコールが同じアドレス（通常の転送）か異なるアドレス間（回線をまたいだ直接転送）かどうかに関係なく、同じイベントが配信されます。この機能は、CTI ポート、SCCP デバイス、SIP デバイスなど、サポートされるすべての電話でサポートされています。

JTAPI API から直接転送が試みられている 2 つのアドレスのいずれかにオブザーバが追加されていない場合、Cisco Unified JTAPI は、エラー「Transfer controller is not set and could not find a suitable TerminalConnection」と共に PlatformException をスローします。

使用上のガイドライン

次のポイントでは、アプリケーションで回線をまたいで直接転送（Direct Transfer Across Lines）機能をどのように使用する必要があるかを示します。

- アプリケーションは、直接転送を試みる両方の回線にコール オブザーバを追加する必要があります。
- 以前のバージョンでは、アプリケーションが、両方のコールで共通のアドレスを使用しているかどうか、およびそのアドレスが同じ端末上にあるかどうかを確認することを推奨していました。回線をまたいで直接転送（Direct Transfer Across Lines）では、直接転送が呼び出される 2 つのコール間で、アドレスが共通かを確認する必要はありません。両方のコールがそれぞれ、共通の端末に存在するアドレスを持つことは確認する必要があります。
- Cisco Unified JTAPI は、現在と同様に、同じアドレスのコールの転送と同じ一連のイベントを報告します。アプリケーションは、Transfer() の呼出し後、CiscoTransferEndEv を受け取るまで、これらのコールについて何もする必要はありません。
- 転送がアドレス間で行われるため、アプリケーションでは CiscoTransferStartEv で共通のコントローラを取得しないため、アプリケーション ロジックをアップグレードする必要があります。

イベント フローの比較とコード例

表 3-1 に、イベント フローの詳細とコード例を示します。

表 3-1　転送呼び出しのイベント フローの比較とコード例

同じ回線での転送	回線をまたがった転送
セットアップ	
端末 T1 のアドレス A 端末 T2 のアドレス B1、B2 端末 T3 のアドレス C	端末 T1 のアドレス A 端末 T2 のアドレス B1、B2 端末 T3 のアドレス C
機能の呼び出し	

表 3-1　転送呼び出しのイベント フローの比較とコード例 （続き）

同じ回線での転送	回線をまたがった転送
A が B1 にコール [GC1=GolbalCallID1]	A が B1 にコール [GC1=GolbalCallID1]
GC1：接続 A1-> Conn1	GC1：接続 A->Conn1
GC1：接続 B1->Conn2	GC1：接続 B1->Conn2
B1 が C にコール [GC2=GolbalCallID2]	B2 が C にコール [GC2=GolbalCallID2]
GG2：接続 B1->Conn3	GG2：接続 B2->Conn3
GC2：接続 C->Conn4	GC2：接続 C->Conn4
GC1.transfer(GC2);	GC1.transfer(GC2);
アプリケーションに配信されるイベント（すべての通話者が監視されているものとする）	

表 3-1　転送呼び出しのイベント フローの比較とコード例 （続き）

同じ回線での転送	回線をまたがった転送
GC1:	GC1:
CiscoTransferStartEv	CiscoTransferStartEv
[getTransferControllerAddress() は B1 を返す]	[getTransferControllerAddress() は B1 を返す]
ConnCreatedEv for C	ConnCreatedEv for C
ConnConnectedEv for C	ConnConnectedEv for C
CallCtlConnEstablishedEv for C	CallCtlConnEstablishedEv for C
TermConnCreatedEv for T3(Address C)	TermConnCreatedEv for T3(Address C)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B1
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B1
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B1)
CallCtlTermConnDroppedEv for T2(Address B1)	CallCtlTermConnDroppedEv for T2(Address B1)
CiscoTransferEndEv	CiscoTransferEndEv
GC2:	GC2:
CiscoTransferStartEv	CiscoTransferStartEv
[getTransfeControllerAddress() は B1 を返す]	[getTransferControllerAddress() は B1 を返す]
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B2)
CallCtlTermConnDroppedEv for T2(Addresss B1)	CallCtlTermConnDroppedEv for T2(Addresss B2)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B2
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B2
TermConnDroppedEv for T3(Address C)	TermConnDroppedEv for T3(Address C)
ConnDisconnectedEv for C	ConnDisconnectedEv for C
CallCtlConnDisconnectedEv for C	CallCtlConnDisconnectedEv for C
CallCtlTermConnDroppedEv for T3(Address C)	CallCtlTermConnDroppedEv for T3(Address C)
CiscoTransferEndEv	CiscoTransferEndEv
CallInvalidEv	CallInvalidEv
CallObservationEndedEv	CallObservationEndedEv
（注）　GC2：端末 T2 のアドレス B1 に対する切断イベント	（注）　GC2：端末 T2 のアドレス B2 に対する切断イベント

表 3-1　転送呼び出しのイベント フローの比較とコード例 （続き）

同じ回線での転送	回線をまたがった転送
アプリケーション コード例	

Handle(CiscoCallEv event) {
…..
…..
if (event instanceOf
CiscoTransferStartEv){
CiscoTransferStartEv ev =
(CiscoTransferStartEv)event;
processTransfer(ev);
}
}
processTransfer(CiscoTransferStartEv ev){
CiscoAddress commonAddr =
ev.getTransferControllerAddress();
 	
CiscoCall GC2 = ev.getTransferringCall();
CiscoCall GC1 = ev.getFinalCall();
CiscoConnection droppedConn1 =
findConnection(GC1, controllerAddr);
 	
CiscoConnection droppedConn2 =
findConnection(GC2, controllerAddr);
//Additional App logic to clear
connections.
}
Connection findConnection(CiscoCall GCx,
CiscoAddress addr){
CiscoConnection[] conns =
GCx.getConnections();
for (i=0; i<conns.length; i++){
if conns[i].getAddress().equals(addr)
{
return conns[i];
}
}
}
	Handle(CiscoCallEv event) {
	…..
	…..
	if (event instanceOf
	CiscoTransferStartEv){
	CiscoTransferStartEv ev =
	(CiscoTransferStartEv)event;
	processTransfer(ev);
	}
	}
	processTransfer(CiscoTransferStartEv ev){
	String termName =
	ev.getControllerTreminalName();
	
	CiscoCall GC2 = ev.getTransferringCall();
	CiscoCall GC1 = ev.getFinalCall();
	CiscoConnection droppedConn1 =
	findConnection(GC1, termName);
	
	CiscoConnection droppedConn2 =
	findConnection(GC2, termName);
	//Additional App logic to clear
	connections.
	}
	Connection findConnection(CiscoCall GCx,
	String termName){
	CiscoConnection[] conns =
	GCx.getConnections();
	for (i=0; i<conns.length; i++){
	CiscoTerminalConnection[] termConns =
	conns[i].getTerminalConnections();
	for(j=0; j<termConns.length; j++){
	
	if(termConns[j].getTerminal().getName.equal
	s(termName)
	&& termConns[i].getState()
	!=
	
	TerminalConnection.PASSIVE){
	return
	termConns[i].getConnection();
	}
	}
	}
	}

（注）　アプリケーション ロジックは、共通の transferControllerAddress に基づき、最後のコールとコンサルト コールの両方に commonAddr が あるため、この例では正常に機能します。	（注）　最後のコールとコンサルト コールでコントローラの共通のアドレスはありませんが、コントローラ TerminalName は両方のコントローラ アドレスで同じです。そのため、アプリケーションは、CommonTerminalName を使用して、接続、TerminalConnection、およびコントローラを検出することができます。

（注）　回線をまたいだ接続転送のシナリオでは、説明したイベントと別に、アプリケーションで、別の一時コール GC3 がアクティブになり（CallActiveEv）、転送の完了直後に GC3 がアイドルになる（CallInvalidEv）ことを認識できます。

インターフェイスの変更

「CiscoTransferStartEv」（P.6-343）を参照してください。

メッセージ シーケンス

「回線をまたいで直接転送（Direct Transfer Across Lines）の使用例」（P.A-207）を参照してください。

下位互換性

この機能は下位互換性があります。アプリケーションに下位互換性を提供するため、回線をまたいで転送を可能にするデバイスへの新しい権限のほか、この権限の新しい標準権限と標準ユーザ グループが追加されました。この新しい権限 Standard Supports Connected Xfer/Conf がアプリケーション ユーザに関連付けられている場合にのみ、アプリケーションはこれらのデバイスを制御できます。アプリケーションは、この新しい権限「Standard CTI Allow Control of Phones supporting Connected Xfer and Conf」がアプリケーション ユーザに関連付けられている場合にのみ、これらのデバイスを制御できます。デフォルトでこれらのデバイスは制限付きとして表示され、アプリケーションで JTAPI 7.1.2 以上を使用することを前提とし、アプリケーションがこの機能を処理できるようにアップグレードされ、新しい権限を関連付けている場合にのみ、これらのデバイスを制御できます。アプリケーションで古い JTAPI クライアントを使用している場合、デバイスは制限されませんが、アプリケーションがこれらのデバイス（この機能の手動での起動をサポートしている）を監視しようとすると、JTAPI は例外をスローし、その時点からこれらのデバイスを制限付きとしてマークします。

ただし、アプリケーションは任意のタイプの電話で、既存の JTAPI transfer() API から 回線をまたいで直接転送（Direct Transfer Across Lines）を呼び出すことができ、アプリケーションは、この機能をサポートする場合にのみこの要求を発行することが予想されているため、この動作は制限されません。さらに、回線をまたいで（直接）転送（Direct/Connected Transfer Across Lines）を実行する FarEnd ポイントも制御されないため、アプリケーションに問題が発生する可能性があります。これはつまり、JTAPI が常にすべての電話について回線をまたいで直接転送（Direct Transfer Across Lines）のイベントを報告することを意味します。

古い JTAPI アプリケーションで、回線をまたいで直接転送（Direct Transfer Across Lines）が呼び出されていない（電話で、または JTAPI API を介して）環境で実行した場合、下位互換性の問題はまったく発生しません。ただし、そのような設定で、この機能を使用する場合は、アプリケーションの変更が必要になります。

シスコでは、複数のアプリケーションで同じ端末を制御または監視したり、同時に処理したりしないことを前提としています。アプリケーションでそのように実行する場合は、このアプリケーションのすべてのインスタンスを変更して、この機能をサポートするか、問題を避けるように調整します。そうしないと、アプリケーションの動作が予測できなくなることがあります。たとえば、App1 と App2 が同じ端末またはアドレスを制御または監視している 2 つのアプリケーションで、App1 がこの機能をサポートするように変更された場合、App2 もこの機能をサポートするように変更する必要があります。そうしないと、共通デバイスでの App1 によるこの機能の呼出しが App2 を切断させる可能性があります。

この機能は、ユーザの操作性を拡張するために設計されているため、すべての Cisco Unified JTAPI アプリケーションでこの機能を評価して、サポートし、必要に応じて、古い動作と新しい動作の両方を処理するコード ロジックでアップグレードすることをお勧めします。

回線をまたいで参加（Join Across Lines）または Connected Conference Across Lines

このリリースでは、既存の Join Across Lines サービス パラメータの範囲外の電話 Cisco Unified IP Phone モデルを採用することによって、ユーザの操作性を拡張しています。これらの電話では、この機能が常に有効にされており、これをオフにするサービス パラメータはありません。機能の詳細説明、インターフェイスの変更に関する情報、使用例については、「Join Across Lines with Conference 機能拡張」（P.3-17）を参照してください。

使用上のガイドライン

次のポイントでは、アプリケーションで回線をまたいで参加（Join Across Lines）機能をどのように使用する必要があるかを示します。

- アプリケーションは回線をまたいで、または接続された会議に参加しようとする両方の回線にコール オブザーバを追加する必要があります。
- 以前のバージョンでは、アプリケーションが、両方のコールで共通のアドレスを使用しているかどうか、およびその共通のアドレスが同じ端末上にあるかどうかを確認することを推奨しました。回線をまたいで参加（Join Across Lines）では、直接会議が呼び出される 2 つのコール間で、アドレスが共通であるかどうかを確認する必要はありません。両方のコールがそれぞれ、共通の端末に存在するアドレスを持つことは確認する必要があります。
- Cisco Unified JTAPI は、現在と同様に、同じアドレスのコールの会議の開始についてと同じ一連のイベントを報告します。アプリケーションは、Conference() の呼出し後、CiscoConferenceEndEv を受け取るまで、これらのコールについて何もする必要はありません。
- 会議がアドレス間で行われる場合は、アプリケーションは CiscoConferenceStartEv で共通のコントローラを取得しないため、アプリケーション ロジックをアップグレードする必要があります。詳細については表 3-2 を参照してください。

イベント フローの比較とコード例

表 3-2 に、イベント フローの詳細とコード例を示します。

表 3-2　会議呼び出しのイベント フローの比較とコード例

同じ回線で参加	回線をまたいで参加
セットアップ	
端末 T1 のアドレス A 端末 T2 のアドレス B1、B2 端末 T3 のアドレス C	端末 T1 のアドレス A 端末 T2 のアドレス B1、B2 端末 T3 のアドレス C
機能の呼び出し	

表 3-2　会議呼び出しのイベント フローの比較とコード例 （続き）

同じ回線で参加	回線をまたいで参加
A が B1 にコール [GC1=GolbalCallID1]	A が B1 にコール[GC1=GolbalCallID1]
GC1：接続 A1' Conn1	GC1：接続 A' Conn1
GC1：接続 B1' Conn2	GC1：接続 B1' Conn2
B1 が C にコール [GC2=GolbalCallID2]	B2 が C にコール [GC2=GolbalCallID2]
GG2：接続 B1' Conn3	GG2：接続 B2' Conn3
GC2：接続 C' Conn4	GC2：接続 C' Conn4
GC1.conference(GC2);	GC1.conference(GC2);
アプリケーションに配信されるイベント（すべての通話者が監視されているものとする）	
GC1:	GC1:
CiscoConferenceStartEv	CiscoConferenceStartEv
[getConferenceControllerAddress() は B1 を返す]	[getConferenceControllerAddress() は B1 を返す]
ConnCreatedEv for C	ConnCreatedEv for C
ConnConnectedEv for C	ConnConnectedEv for C
CallCtlConnEstablishedEv for C	CallCtlConnEstablishedEv for C
TermConnCreatedEv for T3(Address C)	TermConnCreatedEv for T3(Address C)
CiscoConferenceEndEv	CiscoConferenceEndEv
GC2:	GC2:
CiscoConferenceStartEv	CiscoConferenceStartEv
[getConferenceControllerAddress() は B1 を返す]	[getConferenceControllerAddress() は B1 を返す]
TermConnDroppedEv for T2(Address B1)	TermConnDroppedEv for T2(Address B2)
CallCtlTermConnDroppedEv for T2(Addresss B1)	CallCtlTermConnDroppedEv for T2(Addresss B2)
ConnDisconnectedEv for B1	ConnDisconnectedEv for B2
CallCtlConnDisconnectedEv for B1	CallCtlConnDisconnectedEv for B2
TermConnDroppedEv for T3(Address C)	TermConnDroppedEv for T3(Address C)
ConnDisconnectedEv for C	ConnDisconnectedEv for C
CallCtlConnDisconnectedEv for C	CallCtlConnDisconnectedEv for C
CallCtlTermConnDroppedEv for T3(Address C)	CallCtlTermConnDroppedEv for T3(Address C)
CiscoConferenceEndEv	CiscoConferenceEndEv
CallInvalidEv	CallInvalidEv
CallObservationEndedEv	CallObservationEndedEv
（注）　GC2：端末 T2 のアドレス B1 に対する切断イベント	（注）　GC2：端末 T2 のアドレス B2 に対する切断イベント
アプリケーション コード例	

表 3-2　会議呼び出しのイベント フローの比較とコード例 （続き）

同じ回線で参加	回線をまたいで参加

Handle(CiscoCallEv event) {
…..
…..
if (event instanceOf
CiscoConferenceStartEv){
CiscoConferenceStartEv ev =
(CiscoConferenceStartEv)event;
processConference(ev);
}
}
processConference(CiscoConferenceStartEv
ev){
CiscoAddress controllerAddr =
ev.getConferenceControllerAddress();
 	
CiscoCall[] consultCalls =
ev.getConferenedCalls();
CiscoCall GC1 = ev.getFinalCall();
CiscoConnection[] movedConns[] =
findConnections(consultCalls,
controllerAddr);
 	
//Additional App logic to clear
connections.
}
Connection[] findConnections(CiscoCall[]
calls, CiscoAddress addr){
ArrayList connList = new ArrayList();
for(x=0; x < calls.length; x++){
CiscoConnection[] conns =
calls[x].getConnections();
for (i=0; i<conns.length; i++){
if conns[i].getAddress().equals(addr)
{
connList.add(conns[i]);
}
}
}
return connList.toArray(Connection[]
conns);
}
	Handle(CiscoCallEv event) {
	…..
	…..
	if (event instanceOf
	CiscoConferenceStartEv){
	CiscoConferenceEv ev =
	(CiscoConferenceStartEv)event;
	processConference(ev);
	}
	}
	processConference(CiscoConferenceStartEv
	ev){
	String controllerTermName =
	ev.getControllerTreminalName();
	CiscoCall[] consultCalls =
	ev.getConferenedCalls();
	CiscoCall GC1 = ev.getFinalCall();
	CiscoConnection[] movedConns =
	findConnections(consultCalls,
	controllerTermName);
	
	//Additional App logic to clear
	connections.
	}
	Connection[] findConnections(CiscoCall
	calls, String
	termName){
	ArrayList connList = new ArrayList();
	for(x=0; x < calls.length; x++){
	CiscoConnection[] conns =
	calls[x].getConnections();
	for (i=0; i<conns.length; i++){
	CiscoTerminalConnection[] termConns =
	conns[i].getTerminalConnections();
	for(j=0; j<termConns.length; j++){
	
	if(termConns[j].getTerminal().getName.equal
	s(termName)
	&& termConns[i].getState()
	!=
	
	TerminalConnection.PASSIVE){
	connList.add(conns[i]);
	}
	}
	}
	}
	return connList.toArray(Connection[]
	conns);
	}

（注） アプリケーション ロジックは、共通の transferControllerAddress に基づき、最後のコールとコンサルト コールの両方に commonAddr が 存在するため、この例では正常に機能します。	**（注）** 最後のコールとコンサルト コールでコントローラの共通のアドレスはありませんが、コントローラ TerminalName は両方のコントローラ アドレスで同じです。そのため、アプリケーションは、CommonTerminalName を使用して、接続、TerminalConnection、およびコントローラを検出することができます。

（注）　Connected Conference Across Lines のシナリオでは、説明したイベントと別に、アプリケーションで、別の一時コール GC3 がアクティブになり（CallActiveEv）、会議の完了直後に GC3 がアイドルになる（CallInvalidEv）ことを認識できます。

インターフェイスの変更

「CiscoConferenceStartEv」（P.6-120）を参照してください。

メッセージ シーケンス

「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」（P.A-213）を参照してください。

下位互換性

この機能は下位互換性があります。

この機能は特定のデバイスでオフにできず、Cisco Unified JTAPI はこれらの電話に対し、常に回線をまたいで参加（Join Across Lines）のイベントを報告します。ただし、アプリケーションに下位互換性を提供するため、これらのデバイスを制御し、Connected Conference Across Lines を可能にする新しい権限が追加されています。新しい標準権限「Standard CTI Allow Control of Phones supporting Connected Xfer and conf」と標準ユーザ グループも追加されています。アプリケーションは、JTAPI クライアント 7.1.2 以上を使用していることを前提として、この新しい権限がアプリケーション ユーザに関連付けられている場合にのみ、これらのデバイスを制御できます。そのため、デフォルトで、これらのデバイスは制限付きとして表示されます。この機能を処理するために、アプリケーションをアップグレードし、これらのデバイスを制御するための新しい権限を関連付ける必要があります。アプリケーションで古い JTAPI クライアントを使用している場合、デバイスは制限されませんが、アプリケーションがこれらのデバイス（この機能の手動での起動をサポートしている）を監視しようとすると、JTAPI は例外をスローし、その時点からこれらのデバイスを制限付きとしてマークします。

シスコでは、複数のアプリケーションで同じ端末を制御または監視したり、同時に処理したりしないことを前提としています。アプリケーションでそのように実行する場合は、このアプリケーションのすべてのインスタンスを変更して、この機能をサポートするか、問題を避けるように調整します。そうしないと、アプリケーションの動作が予測できなくなることがあります。たとえば、App1 と App2 が同じ端末またはアドレスを制御または監視している 2 つのアプリケーションで、App1 がこの機能をサポートするように変更された場合、App2 もこの機能をサポートする必要があります。そうしないと、共通デバイスでの App1 によるこの機能の呼出しが App2 を切断させる可能性があります。

この機能は、ユーザの操作性を拡張するために設計されているため、すべての Cisco Unified JTAPI アプリケーションでこの機能を評価して、サポートし、必要に応じて、古い動作と新しい動作の両方を処理するコード ロジックでアップグレードすることをお勧めします。

スワップまたはキャンセルと転送または会議の動作

この機能により、Cisco Unified JTAPI で、サポートされる IP フォンでのスワップおよびキャンセル操作をサポートできます。

スワップ操作を呼び出すと、アクティブなコールが保留になり、保留したコールが取得されます。キャンセル操作を呼び出すと、プライマリ コールとコンサルト コール間のコンサルト関係が破棄されます。これらの操作はサポートされる電話からのみ呼び出すことができます。Cisco Unified JTAPI インターフェイスでは、アプリケーションから、スワップ / キャンセル操作を呼び出すことができません。ユーザが電話の [SWAP（切替）] キーを押すと、JTAPI がアクティブな保留されているコールに CallCtlTermConnHeldEv および CallCtlTermConnTalkingEv を配信し、CiscoFeatureReason.REASON_NORMAL でそれらの状態変更を示します。

キャンセル操作が呼び出され、プライマリ コールとコンサルト コールの関係が破棄されても、Cisco Unified JTAPI は直接転送または参加機能を使用して、転送や会議操作を実行できます。コンサルトの開始後に、ユーザが電話の [CANCEL] キーを押した場合、会議や転送が実行されません。電話の [CANCEL（キャンセル）] キーを押すと、アプリケーションへのキャンセル通知がトリガーされます。Cisco Unified JTAPI によって、キャンセル操作を示す CiscoCallFeatureCancelledEv が送信されます。CiscoCallFeatureCancelledEv.getConsultCall() は以前に作成されたコンサルト コールを返します。

接続転送または会議中にキャンセル操作が実行されると、次のことが発生する可能性があります。

- ユーザがアクティブ コール ソフトキーを選択する前に、[CANCEL（キャンセル）] キーを押す。

  この場合、[Transfer（転送）] キーを押すと、consultCall GC3 が作成され、[CANCEL（キャンセル）] キーを押すと、コンサルト コールとして GC2 と GC3 で CiscoCallFeatureCancelledEv がトリガーされます。

- アクティブ コール ソフトキーを押した後、ただし電話 UI でコールを選択する前に、ユーザが [CANCEL（キャンセル）] キーを押す。

  この場合、電話 UI でアクティブ コール ソフトキーを押すと、consultCall GC3 が IDLE になりますが、他の機能の操作が可能であるため、CANCEL 通知はありません。ただし、ユーザが [CANCEL（キャンセル）] キーを押した場合、コンサルト コールを Null として CiscoCallConsultCancelEv がトリガーされます。

- ユーザがアクティブ コール ソフトキーを押し、コールを選択して、[CANCEL（キャンセル）] を押す。

  この場合、選択したコールは、CiscoCallFeatureCancelledEv で consultCall として返されます。

インターフェイスの変更

「CiscoCallFeatureCancelledEv」（P.6-102）を参照してください。

メッセージ シーケンス

「スワップ / キャンセルおよび転送 / 会議の動作変更」（P.A-221）を参照してください。

下位互換性

この機能は下位互換性があります。

このリリースでは、スワップまたはキャンセル機能が有効にされており、それをオフにするサービス パラメータはありません。つまり、Cisco Unified JTAPI は常に、この機能をサポートする電話のスワップまたはキャンセルのイベントをサポートし、報告することを意味します。

ただし、アプリケーションの下位互換性を提供するため、これらのデバイスの制御を可能にし、スワップまたはキャンセル操作を可能にする新しい権限が追加されています。新しい標準権限 Standard Supports Connected Xfer/Conf と標準ユーザ グループがこの機能の admin ページに追加されています。アプリケーションは、JTAPI クライアント 7.1.2 以上を使用していることを前提として、この新しい権限がアプリケーション ユーザに関連付けられている場合にのみ、これらのデバイスを制御できます。デフォルトでこれらのデバイスは制限付きとして表示され、アプリケーションがこの機能を処理できるようにアップグレードされ、新しい権限を関連付けている場合にのみ、これらのデバイスを制御できます。アプリケーションで古い JTAPI クライアントを使用している場合、デバイスは制限されませんが、アプリケーションがこれらのデバイス（この機能の手動での起動をサポートしている）を監視しようとすると、JTAPI は例外をスローし、その時点からこれらのデバイスを制限付きとしてマークします。

この機能は、ユーザの操作性を拡張するために設計されているため、すべての Cisco Unified JTAPI アプリケーションでこの機能を評価して、サポートし、必要に応じて、古い動作と新しい動作を処理するコード ロジックでアップグレードすることを強くお勧めします。

メッセージ受信インジケータの拡張

メッセージ受信インジケータ（MWI）機能の拡張により、アプリケーションは、拡張メッセージ受信カウントをサポートする電話に次のメッセージ カウントを表示することができます。

- 新しい音声メッセージ（標準および高い優先度を含む）の合計件数
- 以前の音声メッセージ（標準および高い優先度を含む）の合計件数
- 高い優先度の新しい音声メッセージの件数
- 高い優先度の以前の音声メッセージの件数
- 新しい FAX メッセージ（標準および高い優先度を含む）の合計件数
- 古い FAX メッセージ（標準および高い優先度を含む）の合計件数
- 高い優先度の新しい FAX メッセージの件数
- 高い優先度の古い FAX メッセージの件数

拡張 MWI メッセージ概要を提供するために、CiscoAddress JTAPI 拡張として、2 つの新しい API が追加されています。既存の setMessageWaiting API と同様に、一方の API では、監視対象のアドレスに対して概要を設定できます。他方の API では、監視対象のアドレスの設定済みのコーリング サーチスペースに定義されている、監視対象のアドレスで到達可能な任意のアドレスにメッセージ概要を設定することができます。

これらの新しい API は拡張メッセージ カウントをサポートしない電話タイプで使用することもできます。これらの API をサポートされない電話で使用した場合、既存の setMessageWaiting メソッドと同じように動作します。つまり、メッセージ受信インジケータ ランプが点灯または消灯するだけで、カウントは表示されません。

インターフェイスの変更

「CiscoAddress」（P.6-37）を参照してください。

メッセージ シーケンス

「拡張された MWI の使用例」（P.A-214）を参照してください。

下位互換性

この機能は下位互換性があります。既存の setMessageWaiting API は変更されません。新しい拡張 MWI 機能を使用する必要がないアプリケーションでは、MWI ランプを設定するために、これらの API を引き続き使用することができます。

パーク監視と Assisted DPark のサポート

この機能は、パーク要求を呼び出すアプリケーションに、新しいパークの復帰動作を提供します。現在、パークの復帰タイマーが切れると、コールがパーク元のアドレスに戻されます。新しい動作によって、パーク モニタリング復帰タイマーが切れても、コールはパーク DN にパークされたままになります。

この機能により、パーク元のアドレスでパークされているコールのステータス監視も可能になります。新しい電話（6900 シリーズ以降）で既存の CiscoConnection.park() JTAPI API を使用するか、電話自体から直接、コールをパークすると、Cisco Unified JTAPI はパークされているコールの現在のステータスを含む新しいイベント CiscoAddrParkStatusEv を配信します。アプリケーションはパーク元のアドレスに AddressObserver を追加し、このイベントを受け取るフィルタを有効にする必要があります。コールがパークされた後にアプリケーションがオブザーバを追加した場合、CAUSE_SNAPSHOT でイベントが配信されます。新しいイベントのパーク ステータスは次のいずれかである可能性があります。

- Parked：アプリケーションのユーザによってコールがパークされたことを示します。
- Reminder：パークされているコールのパーク モニタリング復帰タイマーが切れたことを示します。
- Retrieved：以前にパークされたコールが取得されたことを示します。
- Abandoned：以前にパークされたコールが取得を待つ間に切断されたことを示します。
- Forwarded：Park Monitoring Forward-No-retrieve タイマーが切れたときに、パークされているコールが設定されている Forwarded No Retrieve Destination に転送されたことを示します。

原因が CAUSE_SNAPSHOT の場合、パーク ステータスは、Parked または Reminder 状態のいずれかになります。

電話がこれらの通知が対象となります。つまり、コールをパークしているデバイスのみがこれらの通知を認識できます（パーク元のデバイスと回線を共有する他のデバイスは同様の通知は受け取りません）。Cisco Unified JTAPI に、これを管理するための CiscoAddrParkStatusEv の getTerminal() インターフェイスが追加されています。このインターフェイスは、これらの通知を受け取ったアドレスの端末を返し、この端末はコールをパークした端末です。

さらに、Cisco Unified JTAPI はこの新しいイベントで、アプリケーションに CiscoCallID も提供します。アプリケーションはこれを使用して、コール オブジェクトを取得できます。ただし、このイベントを受け取った時点で、プロバイダーのドメインにコールが存在しない場合、CiscoCallID.getCall() は Null 値を返すことがあります。

Cisco Unified JTAPI は、アプリケーションへの新しいイベント通知を制御またはフィルタする新しいインターフェイス CiscoAddrEvFilter を提供しています。アプリケーションは、CiscoAddrEvFilter インターフェイスの API getCiscoAddrParkStatusEvFilter() および setCiscoAddrParkStatusEvFilter() によってフィルタ値を取得または設定できます。CiscoAddrEvFilter インターフェイスでフィルタの値を取得し、設定するための 2 つの新しいメソッド getFilter() と setFilter() も CiscoAddress に提供されています。フィルタが有効にされており、CiscoAddress で setFilter() が呼び出されている場合にのみ、アプリケーションは新しいイベント通知 CiscoAddrParkStatusEv を受け取ります。下位互換性を維持するために、デフォルトで、CiscoAddrParkStatusEvFilter のフィルタ値は False にされています。

コールがパークされると、パーク モニタリング復帰タイマーが開始し、期限が切れると Reminder が送られこの後、Park Monitoring Forward No Retrieve Timer が開始します。このタイマーが切れ、未取得時のパーク モニタリング転送の接続先が設定されている場合、コールがその宛先に転送されます。転送先で Connection が作成されると、Connection イベントで新しい CiscoFeatureReason FORWARD_NO_RETRIEVE が配信されます。Forward No Retrieve Destination が設定されていない場合、パークの復帰が発生した場合と同じ理由（CiscoFeatureReason.PARKREMINDER）で、パークコールが元の DN に返送されます。

アプリケーションが CiscoAddress.getAddressCallInfo(Terminal term) を呼び出したときに、返される CiscAddressCallInfo が拡張され、パークされているコール数が含まれるようになりました。これはパークされているコール数を返します。Cisco Unified IP Phone 7900 シリーズ（SIP/SCCP）は、このアドレスによってパークされたコールがある場合でも 0 の値を返します。

この機能は、新しい電話（6900 シリーズ以降）でコールがパークされている場合にのみ適用します。Cisco Unified IP Phone 7900 Series（SIP/ SCCP）でコールがパークされた場合、引き続き既存の動作が行われます。新しい Cisco Unified IP Phone でコールをパークし、Cisco Unified IP Phone 7900 シリーズ（SIP）と回線を共有している場合、新しいパーク監視拡張が機能します。ただし、Cisco Unified IP Phone 7900 シリーズ（SIP または SCCP）でパークした場合、アプリケーションがこれらの回線を監視している場合でも、すべての電話で古いパーク動作が行われます。

ユーザは必要に応じて、パーク モニタリング復帰タイマーを 0 に設定し、Park Monitoring Forward No Retrieve タイマーを既存の Park Reversion Duration タイマーに設定して、新しい Cisco Unified IP Phone（Forward No Retrieve Destination が設定されていない場合）で古い動作を得ることができます。ただし、イベント通知は制御できません。

Unified CM サービス パラメータ ページで、先述のタイマーを設定できます。これらは、Cisco Unified IP Phone の 将来のモデルの SIP バージョンにのみ適用されます。

パーク モニタリング復帰タイマー：このタイマーはコールがパークされるとただちに開始します。これは、ユーザにパークされているコールがあることを通知するまでのコールがパークされている時間です。この範囲は 0 ～ 1200 秒で、デフォルト値は 60 秒です。

Park Monitoring Periodic Reversion タイマー：ユーザがパークされているコールについて通知される頻度。この範囲は 0 ～ 1200 秒で、デフォルト値は 30 秒です。

Park Monitoring Forward No Retrieve タイマー：このタイマーはパーク モニタリング復帰タイマーが期限切れになると開始します。これは、パーク先がパーク元の未取得時のパーク モニタリング転送（FNR）の宛先にリダイレクトされるまでのパーク リマインダ通知が再生される秒数です。この範囲は 30 ～ 1200 秒で、デフォルト値は 300 秒です。

未取得時のパーク モニタリング転送の接続先（Park Monitoring Forward No Retrieve Destination）：Unified CM 回線ページ設定の回線ページで設定できます。

Assisted DPark は、電話で DPark 操作を実行するための代替の 1 手順の方法を提供します。ユーザが新しい電話から Assisted DPark を実行し、アプリケーションがパークされている相手を監視している場合に、Cisco Unified JTAPI は DPark DN の接続イベント（ConnCreatedEv、ConnInProgressEv、および CallCtlConnQueuedEv）に理由 CiscoFeatureReason.REASON_REFER を提供します。現在、DPark が実行されると、アプリケーションは CiscoFeatureReason.REASON_TRANSFER の接続イベントを受け取ります。

インターフェイスの変更

「CiscoAddrParkStatusEv」（P.6-59）を参照してください。

メッセージ シーケンス

「パーク モニタリング サポート」（P.A-261）を参照してください。

下位互換性

パーク監視拡張と Assisted DPark のサポートは下位互換性があります。

新しいパーク復帰動作により、ユーザの操作性が向上し、パークされているコールが可能な限り長く取得可能になります。さらに、配信される新しいイベントによって、パークされているコールのステータスを監視できることによって、パーク機能の操作性も向上しています。

アプリケーションは条件によってフィルタを有効/無効にして、CiscoAddeEvFilter の setCiscoAddrParkStatusEvFilter() API を介してイベントを受け取ることができます。デフォルトでこのフィルタは無効にされているため、下位互換性が維持されます。

アプリケーションで 7.1.2 より古い JTAPI クライアントを使用している場合、デバイスは制限されませんが、アプリケーションがこれらのデバイス（この機能の手動での起動をサポートしている）を監視しようとすると、JTAPI は例外をスローし、その時点からこれらのデバイスを制限付きとしてマークします。

論理パーティション化

管理者はこの機能を使用して、地理的位置を設定し、別の地理的位置の VoIP フォンまたは VoIP PSTN ゲートウェイに直接接続された PSTN ゲートウェイを通過するコールを制限できます。この機能により、単一回線のアナログ電話を使用でき、Telecom Regulatory Authority of India（TRAI）規制との互換性を維持できます。

この機能は、Logical Partitioning Enabled サービス パラメータを使用して無効にできます。これはデフォルトで無効にされています。

インターフェイスの変更

「CiscoJtapiException」（P.6-150）を参照してください。

メッセージ シーケンス

「論理パーティション設定機能の使用例」（P.A-293）を参照してください。

下位互換性

この機能は下位互換性があります。

Component Updater

Component Updater インターフェイスは、アプリケーションで更新ログの場所を指定できるように拡張されました。現在、更新ログは、アプリケーションと同じディレクトリに作成されます。この拡張により、アプリケーションはトレースの場所を指定できます。

インターフェイスの変更

「ComponentUpdater」（P.6-348）を参照してください。

メッセージ シーケンス

「ComponentUpdater 拡張の使用例」（P.A-296）を参照してください。

下位互換性

この機能は下位互換性があります。

Cisco Unified IP Phone 6900 シリーズのサポート

この機能により、Cisco Unified JTAPI アプリケーションは、ロールオーバー モードを有効にして端末を制御できます。ロールオーバー モードでは、同じ DN だが異なるパーティション内の、または異なる DN の複数のアドレスで端末を設定できます。ロールオーバー モードを有効にすると、端末で次に使用可能なアドレスでコンサルト コールを作成できます。Cisco Unified IP Phone 6900 シリーズはロールオーバー モードで設定できます。

ロールオーバー モードをサポートしている電話の新しい権限 Standard CTI Allow Control も導入されており、アプリケーションでロールオーバーが有効にされた端末を制御できます。この新しい動作をサポートするアプリケーションは、コンサルト コールが異なるアドレスで作成される場合に、それらのアプリケーションまたはエンド ユーザにこの権限を含める必要があります。そうしないと、ロールオーバー モードで設定されたすべての端末が制限され、addObserver() 要求に例外がスローされます。

この動作をサポートするアプリケーションは、端末にコール オブザーバを追加するか、端末上のすべてのアドレスにコール オブザーバを追加する必要があります。コンサルト コールは次に使用可能なアドレスで作成されるため、すべてのアドレスにコール オブザーバが追加されていない場合、コンサルト 要求に例外がスローされます。

アプリケーションから会議を正常に実行するために、Cisco Unified IP Phone 6900 シリーズで回線をまたいで参加（Join Across Lines）が有効にされている必要があります。

インターフェイスの変更

「CiscoProviderCapabilities」（P.6-211）および「CiscoProviderCapabilityChangedEv」（P.6-213）を参照してください。

メッセージ シーケンス

「Cisco Unified IP Phone 6900 シリーズのサポート」（P.A-297）を参照してください。

下位互換性

この機能は下位互換性があります。

Join Across Lines with Conference 機能拡張

Join Across Lines with Conference 機能は、次のように拡張されました。

- 異なる回線間での会議のチェーニング。たとえば、アプリケーションから、同じ端末上でそれぞれアドレスの異なる 2 つの電話会議を開催できます。
- アプリケーションは、同じ端末上のアドレスが異なる 2 つのコールで会議を開催できます。
- コントローラ以外を使用して会議に参加者を追加できます。

（注）　回線をまたいで参加（Join Across Lines）機能を無効にするには、Join Across Lines Policy サービス パラメータをオフにします。会議のチェーニング機能と、コントローラ以外が参加者を会議に追加できる機能を無効にするには、Advanced Ad Hoc Conference Enabled および Non-linear Ad Hoc Conference Linking Enabled サービス パラメータを無効にします。

アプリケーションから会議要求が発行されたが、選択したコールとアクティブなコールが会議要求に含まれていない場合、次のような動作が発生します。また、この動作は、ユーザが選択したコールが、会議要求には含まれていないが、結果として開催された会議には含まれていた場合にも適用されます。

- 会議が端末の任意のアドレスのコールによって呼び出された場合は、その端末上のアクティブ コールは、必ず結果として開催された会議に追加されます。B1 と B2 アドレスは同じ端末上にあるとします。
  - A --> B1- GC1
  - C --> B1- GC2
  - D --> B2- GC3（アクティブ コール）

  アプリケーションが GC1.conference (GC2) を呼び出した場合、会議要求に D が含まれていなくても、A-B1-C-D（GC1）の会議が開催されます。

  会議が端末の任意の回線上のコールで開催された場合、その端末上のアクティブな電話会議が、結果として開催される会議に追加されます。この場合、アクティブな電話会議が、存続されるコールとなります（アプリケーションにより指定されたプライマリ コールが電話会議ではない場合）。

  この例では、会議の動作が開始されると、アプリケーションが指定したプライマリ コールはクリアされます。アプリケーションが指定したプライマリ コールが、結果として開催された会議に参加できない場合があります。この場合、会議が終了してもプライマリ コールはクリアされません。
- B1 と B2 のアドレスが同じ端末上にあり、conf1 は、B1 をコントローラとする A-B1-C の会議との電話会議であるとします。
  - B1 --> D – GC1（保留）
  - conf1 – GC2（アクティブ コール）
  - B2 --> E – GC3（保留）

  アプリケーションは GC1.conference(GC2, GC3) を呼び出します。これにより、GC2 が存続するコールとなる A-B1-C-D-E の会議が開催されます。アプリケーションは GC1 をプライマリ コールに指定していましたが、会議の終了後は GC1 は存続しません。

この動作は、通常のコントローラで開催される定例会議にも適用されます。A、B、C、および D が異なる端末上に存在するとします。

- A --> B - GC1
- C --> - GC2
- D --> - GC3（アクティブ コール）

  アプリケーションから GC1.conference (GC2) が要求されます。これにより、GC1 で A-B-C-D の会議が開催されます。D とのダイレクト コールが会議要求に含まれていなくても、D は会議に参加できます。

インターフェイスの変更

インターフェイスの変更はありません。現在のインターフェイスを使用して、同じ端末上の異なるアドレスで電話会議を開催できます。

メッセージ シーケンス

「拡張された回線をまたいで参加（Join Across Lines）」（P.A-11）

下位互換性

この機能は下位互換性があります。

ロケール インフラストラクチャの変更

リリース 7.0(1) から、Cisco Unified JTAPI クライアントのインストールで現在サポートされている言語が削除されます。Cisco Unified JTAPI クライアントのインストールでサポートされている言語は英語だけです。また、Cisco Unified Communications Manager サーバから、JTAPI Preference アプリケーションのロケールを動的に更新する機能も追加されます。JTAPI Preference アプリケーションでは、以前のリリースでサポートされていたすべての言語も継続的にサポートされます。新しい言語の追加およびロケール ファイルの更新のサポートも追加されています。

以前のリリースでは、Cisco Unified JTAPI クライアント インストールと JTAPI Preferences アプリケーションはビルド時にローカライズされ、新しい言語のサポートや既存の言語のロケールの更新に対するサポートは追加されていませんでした。JTAPI クライアントのロケールの更新は、Cisco Unified Communications Manager のメンテナンス リリースで実行されていました。ロケールインフラストラクチャの変更により、JTAPI Preferences アプリケーションのロケール ファイルを動的に更新する機能が追加され、JTAPI クライアントのインストールは、英語でだけ実行可能になります。

JTAPI クライアントのインストールには、Cisco Unified Communications Manager TFTP サーバの IP アドレスが必要です。Preferences アプリケーションのロケール ファイルのダウンロードには、TFTP IP アドレスを使用します。TFTP IP アドレスが入力されていない場合や、入力された IP アドレスに誤りがある場合は、Preferences アプリケーションには英語だけが表示されます。今後、新しいロケールの更新が可能になった場合は、JTAPI Preferences アプリケーションから、利用できる更新と最新のロケール ファイルがユーザに通知されます。

インターフェイスの変更

インターフェイスの変更はありません。

メッセージ シーケンス

「ロケール インフラストラクチャ変更シナリオ」（P.A-147）

下位互換性

この機能は JTAPI アプリケーションの観点からは下位互換性がありますが、JTAPI クライアント インストールの観点からすると、この機能により現在サポートされている言語が削除されます。そのため、この点においては、この機能は下位互換性はありません。

Do Not Disturb–Reject

Do Not Disturb–Reject（DND–R）は、既存の Do Not Disturb（DND; サイレント）機能の拡張機能です。これまで、Cisco Unified Communications Manager と JTAPI では、呼出音をオフにする DND（サイレント）機能だけがサポートされていました。ユーザは DND–Reject を使用してコールを拒否できます。DND–R は、Cisco Unified Communications Manager Administration の電話の設定ウィンドウ、または電話プロファイルの設定ウィンドウから設定できます。

DND–R が有効である場合、コール拒否が有効な端末にコールは表示されません。つまり、そのエンドポイントには、着信コールを示す音声または視覚的な表示は行われません。DND–R を有効にするには、DND のステータスを true に設定し、DND オプションを [Call Reject] に設定します。

FeaturePriority により DND が上書きされます。次のいずれかの値が返されます。

- 1: Normal
- 2: Urgent
- 3: Emergency

このリリースでは、CiscoCall の connect() API に FeaturePriority が導入されました。selectRoute() と redirect() API の FeaturePriority は、以前のリリースからすでにサポートされています。connect() API で機能プライオリティが EMERGENCY に指定されており、宛先の端末で DND–R が有効である場合、宛先端末にコールがあると呼出音が鳴り、DND–R 設定が上書きされます。

端末で DND–R が有効であり、インターコム コールを受信すると、DND–R 設定は上書きされ、コールが表示されます。これは、インターコム コールの機能プライオリティは、必ず 2（URGENT）に設定されているためです。

共用回線以外では、A が B にコールし、端末 B の DND–R が有効である場合、原因が USER_BUSY の CallCtlConnFailedEv が A に配信されます。共有 DN を持つすべての端末上で DND–R が有効である場合、ユーザは同じ動作を受信します。

共用回線の 1 つ以上の端末で DND–R が無効であり、その共用回線にコールが発信されると、Cisco Unified JTAPI により、DND–R が有効に設定されている端末に対して TermConnPassiveEv と CallCtlTermConnInUseEv が配信されます（コールの機能プライオリティが NORMAL に設定されていること仮定した場合）。端末上で DND–R が無効である場合は、コール時に、TermConnPassiveEv と CallCtlTermConnBridgedEv が配信されます。

Cisco Unified Communications Manager Administration の電話の設定ウィンドウ、または電話プロファイルの設定ウィンドウで DND（サイレント）オプションが変更されると、新しいイベントである CiscoTermDNDOptionChangedEv が、端末オブザーバに送信されます。

デフォルトの DND オプションは [Ringer–off] で、ルート ポイントで DND はサポートされません。

インターフェイスの変更

「CiscoTermDNDStatusChangedEv」（P.6-301）、「CiscoCall」（P.6-70）、「CiscoTermEvFilter」（P.6-304）

メッセージ シーケンス

「DND–R」（P.A-118）

下位互換性

この機能は下位互換性があります。この機能を設定すると、アプリケーションは新しいイベントを受信します。新しいイベントは、TerminalEventFilter インターフェイスを使用してフィルタされます（CiscoTermEvFilter）。デフォルトでは、このフィルタは無効に設定されており、新しいイベントは配信されません。

発信側の正規化

発信側の正規化（CPN）は拡張された機能です。この機能を使用すると、着信コール番号が変換または正規化され、（国番号、状態コード、番号タイプ）を含む E.164 形式に変換されます。番号タイプフィールドには、加入者、国内、海外、または不明が表示されます。番号タイプは会議のシナリオではサポートされません。

インターフェイスの変更

この機能により、CiscoCall の新しいメソッド getGlobalizedCallingParty() と、CiscoPartyInfo の新しいメソッド getNumberType() が導入されました。詳細については、「CiscoCall」（P.6-70）と「CiscoPartyInfo」（P.6-196）を参照してください。

メッセージ シーケンス

「発信側の正規化」（P.A-148）

下位互換性

この機能は下位互換性があります。

クリック ツー会議

クリック ツー会議機能では、Instant Messenger（IM）などのアプリケーションで会議に通話者を追加できる、SIP トランクのインターフェイスが提供されます。ユーザは、このようなアプリケーションを使用して、他のユーザを会議に追加したり、ユーザを削除することが可能です。クリック ツー会議機能を使用してユーザを会議に追加すると、ターゲット アドレスにコールが提供されます。ターゲット アドレスの 1 つのアドレスだけが、この最初のコール上に作成されます。これにより、このコールは会議に追加され、ターゲット アドレスに新しい callID が作成されます。さらに、このコールの他のアドレスへの接続は新しいコール上に作成されます。

ここでは、クリック ツー会議機能を使用してアドレスが会議に追加されたときに、その対話を処理するために Cisco Unified JTAPI のインターフェイスの変更について説明します。クリック ツー会議機能を使用すると、コンサルト コールは行われず、Cisco Unified JTAPI アプリケーションでは CiscoConferenceStartEv または CiscoConferenceEndEv が受信されません。

この機能を無効にするには、CallManager のサービス パラメータ「ENABLE CLICK TO CONFERENCE」をオフにします。

インターフェイスの変更

「CiscoFeatureReason」（P.6-143）

メッセージ シーケンス

「クリック ツー会議」（P.A-156）

下位互換性

この機能は下位互換性があります。この機能を設定していない場合、または使用していない場合は、Cisco Unified JTAPI アプリケーションに変化はありません。

エクステンション モビリティのユーザ名ログイン

エクステンションモビリティのログイン ユーザ名機能を使用すると、アプリケーションは CiscoTerminal に提供された API からエクステンションモビリティのログイン ユーザ名を取得できます。

インターフェイスの変更

「CiscoTerminal」（P.6-307）

メッセージ シーケンス

「エクステンション モビリティ ログイン ユーザ名」（P.A-182）

Java ソケット接続のタイムアウト

Java ソケット接続のタイムアウト拡張機能を使用すると、Cisco Unified JTAPI 仕様を使用して秒単位でタイムアウトを設定できるので、プライマリ CTI マネージャでは、CTIManager への接続遅延を防ぐことができます。デフォルトは 15 秒です。

デフォルトの 15 秒がアプリケーションに受け入れられない場合は、JAVA API のデフォルトである 0 が、通常の JAVA ソケット接続 API に設定されます。

値の範囲は、5 ～ 180 秒です。デフォルトのゼロを設定すると、JAVA のソケット接続はタイムアウトしません。

インターフェイスの変更

「CiscoJtapiProperties」（P.6-164）

メッセージ シーケンス

「CiscoJtapiProperties」（P.A-184）

下位互換性

この機能は下位互換性があります。

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()

selectRoute() には、機能プライオリティとコーリング サーチ スペースを配列で持っています。この API では、選択したルートごとに、異なる機能プライオリティとコーリング サーチ スペースが柔軟に提供されます。

インターフェイスの変更

「CiscoRouteSession」（P.6-224）

メッセージ シーケンス

「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()」（P.A-181）

下位互換性

この機能は下位互換性があります。selectRoute() API は、機能はそのままで、オーバーロードされた selectRoute() API と相互運用されます。

コール ピックアップ

コール ピックアップを使用すると、コール ピックアップ グループ内のデバイスは、グループ内の他の電話が鳴っているという警告を受け、そのコールをピックアップできます。つまり、呼出音が鳴っている元のデバイスからコールを取得できます。

この機能拡張では、Cisco Unified JTAPI はコール ピックアップ イベントが発生した端末とアドレスの監視をサポートします。ただし、API がアプリケーションからコール ピックアップを起動できません。これは、Auto Call Pickup サービス パラメータが有効に設定されているときの、API によるコール ピックアップ イベントの処理方法に対する小さな変更点です。このパラメータが無効に設定されているときに、ユーザが [Pickup（ピックアップ）] ソフトキーを押すと、そのユーザの電話機の呼出音が鳴りだし、その後はすべて通常通りの処置が可能です。「Auto Call Pickup」が有効に設定されている場合に、このソフトキーを押すと、電話機の呼出音は鳴らず、そのままコールをピックアップできます。

インターフェイスの変更

インターフェイスの変更はありません。

メッセージ シーケンス

「コール ピックアップ」（P.A-168）

下位互換性

この機能は下位互換性があります。

証明書ダウンロードの API 機能拡張

現在、Cisco Unified JTAPI の証明書のダウンロード API にはセキュリティ上の問題があり、この問題を解決するために、新しい証明書のダウンロード API を提供しています。新しい API では、アプリケーションが証明書のパス フレーズを指定する必要があります。証明書のパス フレーズは、クライアント / サーバ証明書が格納された Java キー ストアの暗号化に使用されます。

古い証明書のダウンロード API の使用は推奨しませんが、この API は、アプリケーションの下位互換性の問題を避けるために、まだ残されています。アプリケーションを新しい API に移行することを強くお勧めします。

また、Cisco Unified JTAPI では、新しい API の deleteCertificate() と deleteSecurityPropertyForInstance() も提供されます。これらをアプリケーションで使用すると、すでにインストールされている証明書を削除できます。証明書の Java キー ストアのパス フレーズを変更する場合は、この API を使用して古い証明書を削除し、新しい証明書をアップロードする必要があります。

JTAPIPreferences UI の [セキュリティ] タブの拡張により、[証明書の削除] と [証明書の更新] という 2 つの新しいボタンが追加されました。[証明書の削除] ボタンを使用すると、ユーザは目的のユーザ名またはインスタンス ID の証明書を削除できます。[証明書の更新] ボタンを使用すると、ユーザは

CAPF サーバから証明書をアップロードできます。証明書のアップロードが成功すると、[証明書の更新] ボックスが更新されて更新済みであることが示され、認証文字列と証明書パス フレーズがクリアされます。証明書の更新操作が失敗した場合は、証明書が前に更新されていない限り、証明書ボックスに [未更新] のステータスが示されます。ユーザとアプリケーションは、証明書の更新を行うたびに証明書パス フレーズを提供する必要があり、Cisco Unified JTAPI ではどのような環境下でもセキュリティ上の理由で証明書パス フレーズは保存されません。パス フレーズのセキュリティを確保し、必要な場合には API を通じてパス フレーズを提供することはアプリケーション側の責任です。

下位互換性

この機能は下位互換性があります。

回線をまたいで参加（Join Across Lines）

このバージョンでこの機能を使用すると、アプリケーションは、同じ端末の異なるアドレス上にある 2 つのコールが参加する会議を開くことができます。また、この機能では、アプリケーションはコントローラ以外を使用して、参加者を会議に追加できます。回線をまたいで参加（Join Across Lines）機能は、SIP を実行する CTI 対応の電話でもサポートされます。

回線をまたいで参加（Join Across Lines）機能は、Join Across Lines Policy サービス パラメータをオフにすることで無効にできます。また、会議のチェーニング機能と、コントローラ以外が参加者を会議に追加できる機能は、「Advanced Ad Hoc Conference Enabled」と「Non-linear Ad Hoc Conference Linking Enabled」サービス パラメータを無効にすることで無効にできます。

インターフェイスの変更

この機能には、インターフェイスの変更はありません。アプリケーションは、現在の会議用インターフェイスを使用して、同じ端末上の異なるアドレスが参加する電話会議を開くことができます。

下位互換性

この機能は下位互換性があります。

エクステンション モビリティのインターコム サポート

Cisco Unified Communication Manager のリリース 6.0(1) では、インターコム機能のサポートが追加されました。インターコム機能には、着信側が音声による一方向の自動応答を受け取る必要があるため、インターコムに共有アドレスを設定できません。ユーザがエクステンション モビリティ（EM）プロファイルを使用してログインした場合、インターコムと共有アドレスとなる可能性があります。そのため、これまでエクステンション モビリティはインターコムでサポートされていませんでした。エクステンション モビリティは幅広く利用されるため、この 1 つの宛先を共有できないというインターコム アドレスの性質は維持すると同時に、エクステンション モビリティによるインターコムのサポートというニーズに対処しています。

この機能には、デフォルトの端末で設定されたインターコム アドレスが必要です。この機能を使用すると、インターコム アドレスを EM プロファイル上で設定できます。EM ユーザが端末に、インターコム アドレスで設定された EM プロファイルを使用してログインを行うと、インターコム アドレスのデフォルトの端末がユーザがログインした端末と同じ場合に限り、インターコム アドレスが使用可能になります。インターコム アドレスが端末に設定されているが、この端末はインターコム アドレスのデフォルトの端末ではない場合、インターコム アドレスは端末に表示されません。この端末が Cisco Unified JTAPI アプリケーションの制御リストに設定されている場合、JTAPI はプロバイダーのドメイン内にインターコム アドレスを作成しません。Cisco Unified JTAPI の観点からは、この機能をサポー

トするための新しいインターフェイスや変更はありません。ただし、この機能によって、インターコム機能がインターコム アドレス上で動作しなくなるいくつかの移行シナリオが発生します。使用例を参照してください。

下位互換性

この機能は下位互換性があります。

録音とサイレント モニタリング

この機能を使用すると、アプリケーションによるコールの録音およびサイレント モニタリングが可能になります。この場合の発信者は、モニタリングのターゲットか録音開始側にコールするか、そこからコールを受信する側のエンド ポイントを表します。モニタリングのターゲットはモニタリングの対象側（エージェント）で、モニタリング側はモニタリングの開始側（スーパーバイザ）です。

録音機能により、アプリケーションは任意のモニタリング対象アドレスの会話を録音できます。3 つの録音設定があります。

- 録音なし
- 自動録音

  システムによって録音セッションが始められ、コールが接続状態になると、設定された録音デバイスにメディアがストリーミングされます。
- アプリケーション制御による録音

  アドレスに対してアプリケーション制御による録音が設定されると、アプリケーションは録音を開始および停止できます。アプリケーションが録音を開始する前に、コールが接続状態で存在している必要があります。

管理者は、回線上にいずれかの録音設定を設定できます。

自動録音またはアプリケーションの要求によりトリガーされた録音セッションが確立されると、システムから録音デバイスに、録音開始側からの音声および発信者からの音声の 2 つの音声ストリームが送信されます。

サイレント モニタリング機能により、アプリケーションは他の 2 者間の会話をライブで傍受できます。モニタリングの開始側は、モニタリングのターゲットおよび発信者のいずれとも通話できません。

モニタリングは、アプリケーションの要求によってだけ開始されます。アプリケーションは、モニタリングする各コールに対し、そのつど、モニタリング要求を送信する必要があります。接続状態にあるコールだけをモニタリングできます。モニタリングの要求が確立されると、モニタリングのターゲットと発信者間の音声ストリームがモニタリングの開始側にストリーミングされます。次のような場合、モニタリングのターゲットはトーンを受信します。

- モニタリングのターゲットがトーンを受信するように設定されている場合、または
- アプリケーションにより、モニタリングの開始時にトーンを鳴らすように要求されている場合

録音とサイレント モニタリングの機能では、セキュリティ保護されているコールはサポートしていません。モニタリングのターゲットと録音開始側の両方で、セキュリティを無効にする必要があります。

Cisco Unified Communications Manager Administration の 2 つのユーザ グループで、録音とサイレント モニタリングの機能がサポートされています。アプリケーションでは、それぞれ Standard CTI Allow Call Recording および Standard CTI Allow Call Monitor ユーザ グループに所属している場合に、コールの録音とモニタリングができます。録音とモニタリングに関するイベントは、すべてのコール オブザーバに配信されます。アプリケーションがこれら 2 つの特殊なグループに所属していない場合でも、これらのイベントを受け取ります。

「Monitor」および「Recording」は予約語なので、システム内の回線の表示名としては設定できません。これ以外の予約語には、「Conference」、「Park Number」、「Barge」および「CBarge」があります。

モニタリング セッションが確立されると、モニタリング開始側の端末オブザーバが Cisco RTP イベントを受信します。サイレント モニタリングのコールではメディアが一方向にしか流れませんが、getMediaConnectionMode() は CiscoMediaConnectionMode.RECEIVE_ONLY の代わりに CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE を返します。CTIPort がモニタリング開始側として使用されている場合、アプリケーションでは CiscoMediaOpenLogicalChannelEv から同じ動作を受信する可能性があることを考慮する必要があります。

モニタリング コール（モニタリング開始側によって使用されているコール）を会議にした場合は、最後のコールはモニタリングのターゲットとの接続がありません。モニタリング開始側が別の通話者をモニタリング コールの会議に追加すると、両者にモニタリング ターゲットと発信者との間の音声が聞こえます。

次のインターフェイスは TermConnEv を拡張し、コール オブサーバに配信されます。共用回線の場合、通話中の TerminalConnection のアドレスまたは端末上のコール オブザーバに、これらのイベントが配信されます。端末の接続が INUSE または BRIDGED 状態だけの場合、アプリケーションはイベントを受信しません。

CiscoTermConnRecordingStartedEv

`CiscoTermConnRecordingStartedEv`
録音の開始を示し、録音開始側のコール オブザーバに配信されます。自動録音設定またはアプリケーションの要求によって録音がトリガーされます。

CiscoTermConnRecordingEndEv

`CiscoTermConnRecordingEndEv`
録音の終了を示し、録音開始側に配信されます。

CiscoTermConnMonitoringStartEv

`CiscoTermConnMonitoringStartEv`
モニタリングの開始を示し、モニタリング ターゲットのコール オブザーバに配信されます。このイベントに対して getMonitorType() を使用すると、モニタリングのタイプを返します。

CiscoTermConnMonitoringEndEv

`CiscoTermConnMonitoringEndEv`
モニタリングの終了を示し、モニタリング ターゲットのコール オブザーバに配信されます。

CiscoTermConnMonitorInitiatorInfoEv

モニタリングの開始側の情報が公開され、モニタリング ターゲットのコール オブザーバに配信されます。このインターフェイスには 1 つのメソッドがあります。

`CiscoMonitorInitiatorInfo getCiscoMonitorInitiatorInfo ()`
モニタリングの開始側の端末名およびアドレスを公開する CiscoMonitorInitiatorInfo を返します。

CiscoTermConnMonitorTargetInfoEv

モニタリングのターゲットの情報が公開され、モニタリング ターゲットのコール オブザーバに配信されます。このインターフェイスには 1 つのメソッドがあります。

`CiscoMonitorInitiatorInfo getCiscoMonitorTargetInfo()`
モニタリングのターゲットの端末名およびアドレスを公開する CiscoMonitorInitiatorInfo を返します。

2 つの新しいエラー コードにより、モニタリングの失敗がアプリケーションに通知されます。

- CTIERR_PRIMARY_CALL_INVALID は、コールがアイドル状態になるか転送されたことが原因でモニタリングの要求が失敗し、例外が発生した場合に、CiscoException.getErrorCode() によって返されます。
- CTIERR_PRIMARY_CALL_STATE_INVALID は、モニタリングを開始できない状態にコールが移行したことが原因でモニタリングの要求が失敗した場合に返されます。

このリリースでは、新しい AddressType、MONITORING_TARGET が導入されています。JTAPI では、モニタリングのターゲット アドレスとして、このタイプのアドレスに Connection を作成します。この値は、CiscoAddress.getType() によって返されます。

下位互換性

この機能は下位互換性があります。この機能が設定されて、アプリケーション制御によるアドレスに対して使用されない限り、アプリケーションが新しいイベントを受信することはありません。この機能を有効にするには、Cisco Unified JTAPI アプリケーション ユーザを Standard CTI Allow Call Recording および Standard CTI Allow Call Monitor ユーザ グループに追加します。

これらのインターフェイスの変更に関する詳細については、次のトピックを参照してください。

- CiscoJtapiException
- CiscoAddress
- CiscoCall
- CiscoMediaTerminal
- CiscoMonitorTargetInfo
- CiscoMonitorInitiatorInfo
- CiscoProvider
- CiscoProviderCapabilities
- CiscoProviderCapabilityChangedEv
- CiscoProviderObserver
- CiscoRecorderInfo
- CiscoTerminalConnection
- CiscoTermConnMonitorInitiatorInfoEv
- CiscoTermConnMonitorTargetInfoEv
- CiscoTermConnRecordingTargetInfoEv

インターコム

インターコム機能を使用すると、あるユーザが別のユーザにコールした際に、着信側がビジーであるかアイドルであるかにかかわらず、コールが発信側から着信側への一方向メディアによって自動的に応答されます。着信側は、電話機のディスプレイの応答ソフトキー（無印のキー）を押すか、または TerminalConnection で提供されている join() JTAPI API を起動して発信者と会話を始めることができます。電話機上で特に設定されたインターコム アドレスだけがインターコム コールを開始できます。電話機に設定されたインターコム アドレスに対し、Cisco Unified JTAPI によって CiscoIntercomAddress という新しいタイプのアドレス オブジェクトが作成されます。アプリケーションは、CiscoProvider の getIntercomAddresses () インターフェイスを呼び出すことによって、プロバイダーのドメインに存在するすべての CiscoIntercomAddresses を取得できます。

インターコム コールは、Cisco Unified JTAPI インターフェイスから CiscoIntercomAddress.ConnectIntercom () インターフェイスをコールして開始できます。アプリケーションは、このインターフェイスにインターコム ターゲット DN を提供します。インターコム ターゲット DN がアプリケーションによって事前に構成または設定されている場合、アプリケーションは CiscoIntercomAddress.getTargetDN() インターフェイスを呼び出してターゲットの DN を取得できます。そうでない場合、アプリケーションは有効なインターコム ターゲットを提供しないとコールできません。

インターコム コールはインターコム ターゲットで自動応答されます。Cisco Unified JTAPI により、インターコム ターゲットの TerminalConnection/CallCtlTerminalConnection が Passive/Bridged 状態に移行されます。インターコム ターゲットが応答を開始するには、インターコム ターゲットの TerminalConnection に対してアプリケーションが join () インターフェイスを呼び出します。join () が成功すると、インターコム ターゲットの TerminalConnection/CallCtlTerminalConnection は Active/Talking 状態に移行します。インターコム コールの場合、Cisco Unified JTAPI は次のインターフェイスだけをサポートしています。

- Call.drop ()
- Connection.disconnect ()
- CallCtlTerminalConnection.join ()

アプリケーションは、インターコム コールに対して機能操作を実行できません。CiscoIntercomAddress での接続に対してリダイレクト、コンサルト、転送、会議またはパークを呼び出すと、Cisco Unified JTAPI によって例外がスローされます。また、アプリケーションが CiscoIntercomAddress に対して setForwarding ()、getForwarding ()、cancelForwarding ()、unPark ()、setRingerStatus ()、setMessageWaiting ()、getMessageWaiting ()、setAutoAcceptStatus () または getAutoAcceptStatus () を呼び出すと、例外を受け取ります。

アプリケーションは、提供された API から、設定されたインターコム ターゲットの DN の値および CiscoIntercomAddress のラベルを取得できます。Cisco Unified JTAPI では、デフォルトを返す API と、インターコム ターゲットに現在設定されている値を返す API の 2 種類の API を提供しています。デフォルト値は、Cisco Unified Communications Manager Administration によって事前に設定されているインターコム ターゲットの DN およびラベルです。現在の値は、アプリケーションによって設定されている暫定ターゲットの DN およびラベルです。アプリケーションによって値が設定されていない場合、現在の値はデフォルト値と同じままです。アプリケーションは、CiscoIntercomAddress に対して API の setIntercomTarget () を呼び出して、インターコム ターゲットの DN、ラベルおよび Unicode ラベルを設定できます。インターコム アドレスに対してインターコム ターゲット、ラベルおよび Unicode ラベルを設定できるのは、1 つのアプリケーションだけです。2 つのアプリケーションが値を設定しようとすると、最初のアプリケーションが成功し、2 番目は例外を受信します。インターコム ターゲットの DN とラベルが変更されると、Cisco Unified JTAPI は CiscoIntercomAddress に追加された AddressObserver に対して CiscoAddressIntercomInfoChangedEv を提供します。アプリケーションがインターコム ターゲットの DN およびラベルを設定しており、JTAPI または CTI でフェールオーバーまたはフェールバックが発生すると、JTAPI または CTI は、以前に設定されたインターコム ターゲットの DN、ラベルおよび Unicode ラベルを復元します。JTAPI または CTI がインターコム ターゲットの DN、ラベルまたは Unicode ラベルを復元できない場合、Cisco Unified JTAPI によって CiscoIntercomAddress の AddressObserver に CiscoAddrIntercomInfoRestorationFailedEv が送信されます。アプリケーションが失敗するか、何らかの理由でアプリケーションが停止すると、ターゲットの DN、ラベルおよび Unicode ラベルはデフォルトにリセットされます。JTAPI は、CiscoIntercomAddress に対して resetIntercomTarget () インターフェイスを提供して、インターコム ターゲットをリセットします。

自動応答は、CiscoIntercomAddress に対して常に有効になります。アプリケーションから CiscoAddress に対して getAutoAnswerEnabled () メソッドを呼び出すと、特定のアドレスの自動応答機能を取得できます。

インターコムの開始側に一方向メディアで接続されているインターコム ターゲットの場合、デバイスの状態は CiscoTermDeviceStateWhisper に設定されます。これは、端末オブジェクトの新しいデバイス状態です。この状態では、端末は新しいコールを開始したり、新しい着信コールを受けられます。アプリケーションがこのデバイス状態の受信に対してフィルタを有効にしている場合、アプリケーションは CiscoTermDeviceStateWhisperEv を受信します。アプリケーションは、CiscoTermEvFilter に対して setDeviceStateWhisperEvFilter() をコールしてフィルタを有効にできます。DeviceStates の DEVICESTATE_ACTIVE、DEVICESTATE_HELD、DEVICESTATE_ALERTING は、いずれも、DEVICESTATE_WHISPER を無効にします。もし 1 つのコールがアクティブ、保留またはアラート状態で、別のコールが whisper の場合、DeviceState はそれぞれ DEVICESTATE_ACTIVE、DEVICESTATE_HELD または DEVICESTATE_ALERTING になります。

（注） Cisco Unified JTAPI では、インターコム ターゲットが開始側に応答するために、javax.telephony.TerminalConnection インターフェイスの join() を実装しています。システムでは、CiscoIntercomAddresses に対してのみこのインターフェイスを実装しています。Passive または Bridged 状態の通常の共用回線 … 対してアプリケーションがこのインターフェイスを呼び出すと、JTAPI によって MethodNotImplimented 例外がスローされます。

ヒント アプリケーションで制御されているデバイス（端末）にインターコム回線が設定されていない場合、この機能には下位互換性があります。アプリケーションは、アプリケーションで制御されているデバイス（端末）にインターコム回線を設定しないことにより、インターコム機能を無効にできます。

これらのインターフェイスの変更に関する詳細については、次のトピックを参照してください。

- CiscoIntercomAddress
- CiscoAddrIntercomInfoRestorationFailedEv
- CiscoAddress
- CiscoCall
- CiscoProvider
- CiscoTermEvFilter
- CiscoTerminal
- CiscoTerminalConnection
- CiscoTermDeviceStateWhisperEv

アラビア語とヘブライ語の言語サポート

このバージョンの Cisco Unified JTAPI では、アラビア語とヘブライ語をサポートしています。これらの言語は、インストール時および Cisco Unified JTAPI Preferences のユーザ インターフェイスで選択できます。

下位互換性

この機能は下位互換性があります。

Do Not Disturb（サイレント）

Do Not Disturb（DND; サイレント）機能を使用すると、電話機のユーザが、電話機を離れるときや受信コールに応答したくない場合に、電話機を DND 状態にできます。[DND] ソフトキーを使用して、この機能を有効または無効に設定します。

ユーザ ウィンドウから、DND の次の設定を指定できます。

- DND Option：Ringer オフ
- DND Incoming Call Alert：ビープのみ / フラッシュのみ / 無効
- DND Timer：0 ～ 120 分の値。「DND（サイレント）がアクティブであることをユーザに知らせる時間（分単位）」を指定します。
- DND status：on/off

（注）　アプリケーションからは、DND ステータスを有効または無効に設定することだけが可能です。

- アプリケーションから DND ステータスを設定するには、CiscoTerminal で新しいインターフェイスを呼び出します。
- DND ステータスが電話機、Cisco Unified Communications Manager Administration、またはアプリケーションによって設定されると、JTAPI も DND ステータスの変更をアプリケーションに照会します。
- アプリケーションでは、事前の通知を受信するためには、CiscoTermEvFilter のフィルタを有効にする必要があります。
- アプリケーションでは、CiscoTerminal の新しいインターフェイスから、DND ステータスも照会できます。
- アプリケーションでは、CiscoTerminal の新しいインターフェイスから、DND オプションも照会できます。

（注）　この機能は、電話機と CTI ポートに適用できます。ルート ポイントは該当しません。

CER アプリケーションが発信した緊急コールが、DND が有効になっているアプリケーションに着信すると、DND 設定が無効にされ、コールがアプリケーションに繋がれます。CiscoCall、CiscoConnection および CiscoRouteSession の redirect() および selectRoute() API に含まれる FeaturePriority という新しいパラメータにより、この機能が提供されます。緊急コールを発信する CER アプリケーションにより、FeaturePriority が FeaturePriority_Emergency に設定されます。アプリケーションが feature priority を設定するのは、緊急コールだけです。通常のコールの場合、アプリケーションは feature priority をまったく設定しないか、または FeaturePriority_Normal に設定します。通常のコールで、アプリケーションが FeaturePriority_Emergency に設定することはありません。インターコムなどの機能コールを発信する場合、アプリケーションは FEATUREPRIORITY_URGENT を設定する必要があります。

（注）　CiscoCall の connect() API では FeaturePriority パラメータをサポートしていません。

デバイスがイン サービスになる前に getDNDStatus()、setDNDStatus() または getDNDOption() を実行すると、アプリケーションに例外が返されます。

setDNDStatus() 要求の送信後に発生した DB アップデートの失敗またはデバイスのアウト オブ サービスの状況に対処するために、DND（サイレント）に事後条件が追加されます。setDNDStatus() 要求の送信後に DB アップデートの失敗またはデバイスのアウト オブ サービスの状況が発生すると、setDNDStatus() によってアプリケーションに CiscoTermDNDStatusChangedEv が配信されます。このイベントを受信しなかった場合、事後条件によるタイムアウトが発生し、「could not meet post conditions of setDNDStatus()」という例外がスローされます。

下位互換性

この機能は下位互換性があります。この機能を設定すると、アプリケーションは新しいイベントを認識します。新しいイベントは、TerminalEventFilter インターフェイスからフィルタできます（CiscoTermEvFilter）。デフォルトで、このフィルタは無効に設定されており、システムは新しいイベントを配信しません。

詳細は、次のトピックを参照してください。

- CiscoTerminal
- CiscoTermDNDStatusChangedEv
- CiscoTermEvFilter
- CiscoCall
- CiscoConnection
- CiscoRouteSession
- CiscoTermInServiceEv

セキュア会議

この機能では、コールがセキュアかどうかアプリケーションに通知され、セキュアな電話会議を可能にします。コール全体のセキュリティ ステータスが変更されると、セキュア会議からは、コールでのイベントという形でアプリケーションに通知されます。コール全体のセキュリティ ステータスが変更されると、CiscoCallSecurityStatusChangedEv で全体的なコールのセキュリティ ステータスを受信します。端末が通話状態になると、JTAPI により、コールのセキュリティ ステータス情報がアプリケーションに配信されます。アプリケーションは、CiscoCall の新しいインターフェイスを使用して、コールのセキュリティ ステータスを照会できます。アプリケーションが既存のコールのモニタリングを始めると、アプリケーションではセキュリティ ステータス情報を参照できるようになります。

共有アドレスの場合、CiscoCallSecurityStatusChangedEv も Remote In Use の状態のアドレスにも報告されます。OverallCallSecurityStatus は、アクティブな端末で報告されるステータスと一致します。たとえば、A（暗号化）、B（暗号化）、C（認証済）および C'（認証済）という三者会議の場合、システムにより、C および C' に CiscoCallSecurityStatusChangedEv with OverallCallSecurityStatus = Authenticated が報告されます。このイベントは、コールごとに配信されます。

OverallCallSecurityStatus が Encrypted であるかどうかに関係なく、SRTP キー情報は引き続き暗号化されている通話者について送信されます。たとえば、A（暗号化）、B（暗号化）および C（セキュアでない）という三者会議の場合、電話会議の OverallCallSecurityStatus は NotAuthenticated です。ただし、暗号化されている通話者が含まれているため、A、B および会議ブリッジを繋ぐメディアは暗号化されたままです。したがって、OverallCallSecurityStatus に関係なく、A と B は SRTP キーを受信します。

下位互換性

この機能は下位互換性があります。jtapi.ini ファイルの新しいパラメータ EnableSecurityStatusChangedEv が、セキュア会議の機能によって生成される新しいイベント CiscoCallSecurityStatusChangedEv を制御します。アプリケーションでは、jtapi.ini ファイルに

「EnableSecurityStatusChangedEv=1」という行を追加することにより、このパラメータをオンにしてこの新しいイベントを受信できます。デフォルトで、このパラメータは jtapi.ini ファイルに含まれていないため、イベント通知は無効です。com.cisco.jtapi.extensions.CiscoJtapiProperties の setCallSecurityStatusChangedEv() インターフェイスにより、アプリケーションでこの ini パラメータをプログラムから設定できます。

詳細は、CiscoCallSecurityStatusChangedEv を参照してください。

Cisco Unified IP 7931G フォンの対話

Cisco Unified IP 7931G フォンは、次の 2 つのモードに設定できます。

- NoRollOver
- RollOver（同じ DN 間または異なる DN 間で）

Cisco Unified IP 7931G フォンが NoRollOver モードに設定されると、SCCP を実行する通常の電話と同様に機能します。このモードでは、異なるアドレス間で転送や会議を実行できません。JTAPI は、7931G フォンが NoRollOver モードに設定されている場合、制御およびモニタリングをサポートします。

RollOver モードでは、Cisco Unified IP 7931G フォンは、異なるアドレス間での転送や会議をサポートします。このモードでは、JTAPI は、Cisco Unified IP 7931G フォンの制御およびモニタリングができません。アプリケーションでは、このような端末やアドレスは制限されていると認識されます。Cisco Unified IP 7931G フォンがアプリケーション ユーザの制御リストに含まれており、設定が NoRollOver から RollOver モードに変更されると、JTAPI は、原因 CiscoRestrictedEv.CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION で、CiscoAddrRestrictedEv イベントを Cisco Unified IP 7931G フォンの各アドレスに送信し、CiscoTermRestrictedEv を各端末に送信します。

ただし、電話の設定が RollOver から NoRollOver モードに変更されると、JTAPI は、CiscoAddrActivatedEv イベントを Cisco Unified IP 7931G フォンの各アドレスに送信し、CiscoTermActivatedEv を各端末に送信します。

RollOver モードに設定されている Cisco Unified IP 7931G フォンが JTAPI によって制御されたアドレスに転送または会議を発信すると、JTAPI のアプリケーションは最後のコールとコンサルト コールで共通のコントローラを認識しません。これにより、JTAPI アプリケーションの動作が異なってきます。JTAPI アプリケーションがイベントの情報を処理している方法によっては、この転送または会議の場合に JTAPI イベントを処理する方法を変更する必要があります。

異なるアドレス間での転送および会議を無効にするには、Cisco Unified Communications Manager Administration の phone configuration ウィンドウで、Cisco Unified IP 7931G フォンを NoRollOver（ロールオーバーなし）モードに設定します。

CiscoRestrictedEv インターフェイスには、2 つの新しい原因コードがあります。Cisco Unified IP 7931G フォンが RollOverMode に設定されているために端末またはアドレスが制限されている場合、JTAPI は原因 CiscoRestrictedEv.UNSUPPORTED_DEVICE_CONFIGURATION で CiscoAddrRestrictedEv を送信します。また、このリリースではデフォルトの原因コード CAUSE_UNKNOWN も導入されており、これはアプリケーションが処理可能です。

下位互換性

この機能は下位互換性があります。この機能を無効にするには、クラスタ内のすべての Cisco Unified IP 7931G フォンを NoRollOver モードに設定するか、または Cisco Unified Communications Manager のクラスタ内に Cisco Unified IP 7931G フォンを含めないようにします。Cisco Unified Communications Manager のクラスタ内の電話が 1 つでも RollOver モードに設定されている場合、JTAPI が制御するアドレスや端末の動作に変更を及ぼす可能性があります。

詳細は、CiscoRestrictedEv を参照してください。

バージョン形式の変更

リリース 6.0 では、Cisco Unified JTAPI のバージョンが 4 桁の形式から 5 桁の形式に変更になります。これは、Cisco Unified Communications Manager で使用されている形式と同じです。JTAPI のバージョンは Cisco Unified Communications Manager のバージョンと同じままです。新しいインターフェイスでは、アプリケーションは拡張されたバージョン番号を取得します。CiscoJtapiVersion を参照してください。

下位互換性

この機能は下位互換性があります。

発信側の IP アドレス

CallCtlConnOfferedEv および RouteEvent の拡張により、発信側の IP アドレス取得の手段が提供されます。この機能は、基本的な通話、転送や会議のためのコンサルト コール、および基本的なリダイレクトと転送の着信側に、発信側の IP アドレスを通知します。発信側が変更になる場合を含め、これ以外の状況や機能対話はサポートされていません。この機能では発信側のデバイスとして IP フォンだけをサポートしていますが、他の発信デバイスの IP アドレスも提供できます。
CiscoCallCtlConnOfferedEv および CiscoRouteEvent を参照してください。

下位互換性

この機能は下位互換性があります。

MLPP（Multilevel Precedence and Preemption）のサポート

Cisco Unified Communications Manager では、MLPP（Multilevel Precedence and Preemption）用に設定された電話機による補足サービスの使用が可能です。Cisco Unified Communications Manager は、これを実現するために、コールの優先順位レベルを設定しています。

（注）　JTAPI では、アプリケーションの優先順位レベルを提供していません。

コントローラ以外による会議への通話者の追加

会議に参加しているすべての通話者が、会議に参加者を追加できるようになりました。以前のリリースでは、参加者を追加できるのは会議コントローラだけでした。

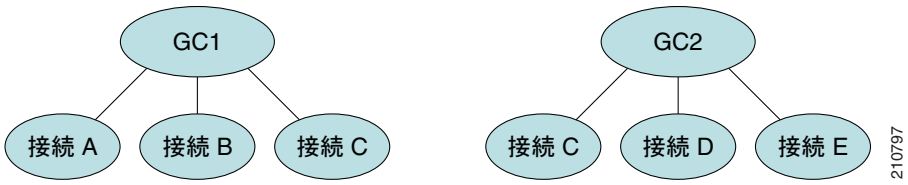
- CiscoConferenceStartEv には、要求者の識別子が含まれています。
- メソッド getConferenceControllerAddress は、要求者の Terminal ConnectionTerminalConnection を返します。
- CiscoConferenceStartEv の新しいメソッド getOriginalConferenceControllerAddress() は、元のコントローラの TerminalConnectionTerminal Connection を返します。

会議のチェーニング

会議のチェーニング機能を使用すると、アプリケーションが 2 つの別々の電話会議を繋げることができます。JTAPI アプリケーションでは、チェーニングされた電話会議は 2 つの別々のコールとして認識されます。電話会議がチェーニングされると、JTAPI によって会議チェーン用に新しい Connection が作成され、CallCtlCallObserver に CiscoConferenceChainAddedEv イベントが提供されます。会議のチェーンがコールから削除されると、JTAPI によって会議チェーンの接続が解除され、CallCtlCallObserver に CiscoConferenceChainRemovedEv イベントが提供されます。アプリケーションは、CiscoConferenceChainAdded/RemovedEv から、すべての会議チェーン接続へのリンクを提供する CiscoConferenceChain を取得できます。

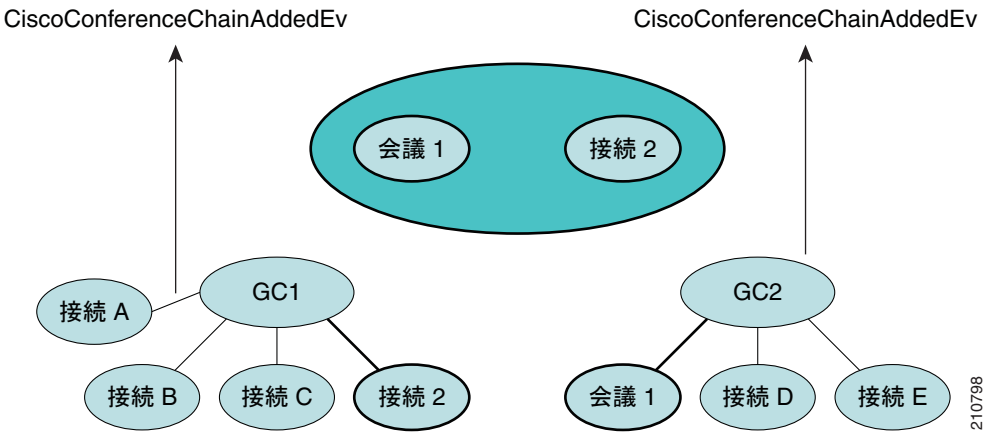
図 3-1 は、電話会議 GC1 に参加している通話者 A、B および C と、電話会議 GC2 に参加している通話者 C、D および E を示しています。

図 3-1　　チェーニング前のコール



会議チェーンが作成された後、コールは図 3-2 のようになります。

図 3-2　　チェーニング後のコール



アプリケーションは、チェーニングされた会議のすべての参加者を CiscoChainedConference オブジェクトから取得できます。このオブジェクトは、チェーニングされているすべての電話会議の会議チェーン Connection を返します。Connection のリストを参照することにより、すべてのチェーニングされた電話会議のリストを取得できます。ただし、各会議から最低 1 人の参加者がアプリケーションで監視されている必要があります。

（注） 会議のチェーニング、またはチェーニングされた電話会議への参加者の追加が関わる会議のシナリオでは、JTAPI は ConferenceStarted/Ended イベントを提供しません。

詳細は、次のトピックを参照してください。

- CiscoCall（getConferenceChain() インターフェイス用）

- CiscoConferenceChain
- CiscoConferenceChainAddedEv
- CiscoConferenceChainRemovedEv

帯域幅不足および未登録 DN 発生時の転送

この機能では、帯域幅不足および未登録 DN 発生時に対処するために、転送ロジックが次のように拡張されています。

- 帯域幅不足のためにコールをリモートの宛先に配信できない場合、コールは AAR 宛先マスクまたはボイスメールに再ルーティングされます。これらの設定は、Cisco Unified Communications Manager GUI の電話番号ウィンドウでユーザが変更できます。
- 未登録 DN：コールが未登録 DN に対して発信されると、コールは Call Forward on No Answer（CFNA; 無応答時コール転送）に設定されている DN に配信されます。

Call Forward No Bandwidth（CFNB）のためにコールが QSIG を使用するトランク / ゲートウェイを越えて別の着信先クラスタに転送された場合、コールの履歴が失われる可能性があります。たとえば、電話 A から帯域幅の少ない地域にある電話 B をコールし、CFNB でコールを別のクラスタにある電話 C に転送するように設定されている場合、クラスタ間の転送に QSIG 使用されていると、元の着信側と最後にリダイレクトした通話者は着信先に伝達されない可能性があります。

ダイレクト コール パーク

この機能では、ユーザが自分の選択したパーク コードにコールを転送して、コールをパークできます。

例

A が B にコールし、B がそのコールをパーク DN に転送する。転送が完了すると、A から B へのコールは指定済みのパーク DN でパークされる。A には MOH が再生される（設定されている場合）。C が（プレフィクス コードとパーク コードをダイヤルして）コールをパーク解除すると、A と C が接続される。

A が パーク DN に直接コールする場合は、A はパーク DN に接続され、このパーク DN はビジーとしてマークされる。A はパークの復帰まで、このパークされた DN に接続されたままになる。

C がパーク DN でコールのパークを解除しない場合は、コールをパークした DN（B）に、このコールのパークが復帰して A と B が再び接続されます。B は別のパーク DN に d-Park を再試行することができます。パークの復帰が発生すると、Cisco Unified Communications Manager JTAPI が新しい理由コードをアプリケーションに渡します。

CTI は、「パーク番号 : (<プレフィクス コード>)<DPark DN>」の形式でパークされた番号を Cisco Unified Communications Manager JTAPI に送信します。Cisco Unified Communications Manager JTAPI はこの情報を解析し、プレフィクス コードと DPark DN の両方をアプリケーションに公開します。

コールのパークが解除されると、パークされていた通話者とパークを解除している通話者の両方が、CTI が付与した理由コードが設定された CPIC イベントを受信し、パークされていた通話者がパークを解除している通話者と接続されます。

通話者 A が dPark DN をコールして、B も同じ dPark DN をコールすると、A か B のどちらかは dPark DN に接続できますが、他方は接続解除されます。

Cisco Unified Communications Manager JTAPI のサポート

Cisco Unified Communications Manager JTAPI では、この機能がサポートされています。ダイレクト コール パーク DN にコールが転送される（dPark される）と、ダイレクト コール パーク DN 用に Connection が作成されてその Call Control Connection のステータスが CallControlConnection.QUEUED になったことがアプリケーションで認識され、CiscoTransferstart イベントと End イベントが配信されます。アプリケーションは、CiscoConnection の新しいインターフェイスを使用して、コールのパークを解除するために必要なプレフィクス コードを取得できます。

パフォーマンスとスケーラビリティ

この機能のパフォーマンスへの影響は、既存の転送機能と同じです。

ボイス メールボックスのサポート

この機能では、ボイス メールボックスの番号が公開されるので、Cisco Unified Communications Manager JTAPI アプリケーションで、電話番号から正しいボイス メールボックスへコールを転送できます。

Cisco Unified Communications Manager の管理者は、各電話番号にボイスメール プロファイルを関連付けることかできます。いずれかの転送設定でボイスメール オプションが有効になっているときに、対応する転送が有効になっている場合は、ボイスメール プロファイルに関連付けられているボイス メール パイロット番号にコールが転送されます。

ボイスメール プロファイルには、ボイスメール パイロット番号フィールドとボイス メールボックス マスク フィールドがあります。ボイス メールボックス マスクでは、自動登録された電話機のボイス メールボックス番号をフォーマットするために使用するマスクを指定します。自動登録された電話機の電話回線からボイス メッセージング システムにコールを転送するときには、Cisco Unified Communications Manager によって、その電話回線の Voice Mail Box フィールドに設定されている番号にこのマスクが適用されます。

たとえば、972813XXXX というマスクを指定した場合、ディレクトリ番号 7253 のボイス メールボックス番号は 9728137253 になります。マスクを入力しない場合は、ボイス メールボックス番号はディレクトリ番号（この例では 7253）と同じ番号になります。

Cisco Unified Communications Manager JTAPI のサポート

この機能をサポートするために、Cisco Unified Communications Manager JTAPI では、called、lastRedirected、originalCalled の各通話者のボイス メールボックス番号が公開されています。これらのボイス メールボックスのフィールドは、CiscoCall オブジェクトで公開されている CiscoPartyInfo で公開されています。通話者にボイスメールが設定されていない場合、Cisco Unified Communications Manager JTAPI は、ボイス メールボックスのフィールドに空の文字列を返します。

Cisco Unified Communications Manager JTAPI の以前のリリースでは、ボイス メールボックスのフィールドをアプリケーションに公開していませんでした。そのため、Cisco Unified Communications Manager JTAPI ボイス メールボックス アプリケーションでは、voicemailboxmask がボイスメール プロファイルに設定されているかどうかが判断できず、電話番号とは異なるボイス メールボックス番号が生成される可能性がありました。

パフォーマンスとスケーラビリティ

この機能により、Cisco Unified Communications Manager JTAPI レイヤからアプリケーション レイヤへのトラフィックが増加することはありません。ただし、ネットワーク上で追加のフィールドが渡されるので、パフォーマンスに多少影響する可能性があります。

Privacy On Hold

この機能は、保留中のプライベート コールのプライバシーを強化するものです。プライバシーが有効になっていると、コールを保留状態にした電話機だけがそのコールを取得でき、発信者の名前と番号は表示されません。

この機能では、共用アドレスで、他の共用アドレスからのコールへの割り込みを許可するかどうかを決定することができます。プライバシーを有効にすると、他の共用アドレスからコールに割り込めなくなります。プライバシーは、端末のプロパティです。IP フォンでは、プライバシー機能ボタンによって、ユーザがプライバシー機能の有効と無効を切り替えることができます。端末のアクティブ コールに対しては、プライバシーの有効と無効を動的に切り替えることができます。コールのプライバシーがオンになっているときには、他の共用アドレスから参照できる TerminalConnection の状態は In Use（使用中）に設定されます。CallProgress 中にプライバシーの状態が変化した場合は、CiscoTermConnPrivacyChangedEvent がアプリケーションに配信されます。

以前のリリースでは、プライバシーが有効の場合にコールが保留状態にされると、すべての TerminalConnection が TermConnHeld 状態になり、他の共用アドレスの任意の terminalConnection からコールの保留を解除できていました。Cisco Unified Communications Manager 4.2 では、Enforce Privacy on Held Calls サービス パラメータが有効になっている場合に、コールのプライバシーが有効になっていると、コールを保留状態にしても他の共有アドレスの terminalConnection は変更されず、In Use（使用中）状態のままになります。

パフォーマンスとスケーラビリティ

この機能では、Cisco Unified JTAPI、アプリケーション、Cisco Unified Communications Manager の間に追加のトラフィックは生成されないので、パフォーマンスへの影響はありません。

Cisco RTP イベントの CiscoRTPHandle インターフェイス

次のインターフェイスは、アプリケーションがイベントから CiscoRTPHandle を取得できるように拡張されました。

- CiscoRTPInputStartedEv
- CiscoRTPInputStoppedEv
- CiscoRTPOutputStartedEv
- CiscoRTPOutputStoppedEv

CiscoRTPHandle は、Cisco Unified Communications Manager のコールの CallID を表し、端末上でコールがアクティブな間は変化しません。コールと関連付けられた GCID が変更されても、特定の端末/アドレス上では CiscoRTPHandle は一定に保たれます。

保留の復帰

保留の復帰機能では、アプリケーションに通知が送信されます。この通知には、保留中のコールの存在を Cisco Unified Communications Manager がアドレスに通知するものと、コールが一定時間 ONHOLD 状態であることを通知するものがあります。アプリケーションは、コールを ONHOLD にしたアドレスのコール オブザーバ上で、この通知を CiscoCallCtlTermConnHeldReversionEv Call ControlTerminalConnection イベントとして受信します。この通知は、保留中のコールについて 1 回だけアプリケーションに送信されます。

イベントは、コールが保留された端末の TerminalConnection にだけ送信されます。アドレスが共用回線のアドレスである場合、共用回線アドレスの他の TerminalConnection はイベントを受信しません。

このイベントを受信するには、アプリケーションはアドレスにコール オブザーバを追加する必要があります。このイベントの原因は CAUSE_NORMAL です。保留復帰タイマーの期限が切れて、通知が電話機に送信された後にコール オブザーバが追加された場合、アプリケーションには原因 CAUSE_SNAPSHOT で CiscoCallCtlTermConnHeldReversionEv が送信されます。

詳細は、CiscoCallCtlTermConnHeldReversionEv を参照してください。

トランスレーション パターンのサポート

JTAPI アプリケーションが制御する Address に適用されているトランスレーション パターンに対して発信者番号変換マスクを設定すると、発信側と着信側の両方を監視している場合、アプリケーションは余分の Connection の作成および Connection 解除を認識できます。着信側だけが監視されている場合、実際の発信者ではなく、トランスフォームされた発信者に対して Connection が作成され、CiscoCall.getCurrentCallingParty() はトランスフォームされた発信者を返します。一般的に、JTAPI は Call 内の適切な Connection を作成できない可能性があり、currentCalling、currentCalled、calling、called および lastRedirecting の通話者について正確な情報を提供できない可能性があります。

たとえば、トランスレーション パターン X で、発呼側トランスフォーメーション マスク Y と着信側トランスフォーメーション マスク B が設定されているとします。A が X をコールすると、コールは B に着信します。この場合、次のようになります。

- アプリケーションが B だけを監視している場合、JTAPI は Y と B に対して Connection を作成し、CiscoCall.getCurrentCallingParty() は Address Y を返します。
- アプリケーションが A と B の両方を監視している場合、A と B に対して Connection が作成され、Y に対して Connection が一時的に作成されて破棄され、CiscoCall.getCurrentCallingParty() は Address Y を返します。

基本的なコールに対して他の機能も実行すると、これ以外にもコールの情報の不統一が発生する可能性があります。JTAPI アプリケーションが制御するアドレスに適用される可能性のあるトランスレーション パターンに対しては、発呼側変換マスクを設定しないことを推奨します。

発信側の IP アドレス

発信側の IP アドレスの拡張機能は、基本的な通話、転送や会議のためのコンサルト コール、および基本的なリダイレクトと転送の着信側に、発信側の IP アドレスを通知します。この機能では発信側のデバイスとして IP フォンだけをサポートしていますが、他の発信デバイスの IP アドレスも提供できます。

（注）　この他の機能対話は、発信側が変更される場合の対話を含め、サポートされていません。

Cisco では、新しく CallCtlConnOfferedEv と RouteEvent クラスに対する機能拡張が行われ、発信側の IP アドレスを取得するためのメソッドが公開されました。この新しい拡張機能は、CiscoCallCtlConnOfferedEv と CiscoRouteEvent です。発信側の IP アドレスが使用できない場合は、空の値が戻ります。

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 アドレスが B を監視する JTAPI で使用可能な場合の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 に通話する

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 に通話する

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

通話者 A が B を通話者 C にリダイレクトする

B を監視する JTAPI アプリケーションでは、発信側の IP アドレスは*使用できない*

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

下位互換性

この機能は下位互換性があります。アプリケーションは、新しい API を起動してコールの IP アドレスを問い合せる必要があります。

回線をまたいで参加（Join Across Lines）

回線をまたいで参加（Join Across Lines）機能を使用すると、回線をまたいで参加をサポートできます。この機能では、電話機の [Join] ソフトキー、または JTAPI が提供する conference() API を使用して、同じ端末にある異なるアドレス上の複数のコールを会議に参加させることができます。アプリケーションは最後のコールとコンサルト コールで共通のコントローラを認識できない場合、JTAPI アプリケーションに対する動作が変わります。

API 自体は変更されず、会議に参加するコールが同じアドレス（通常の会議）か異なるアドレス間（回線をまたいで参加）かどうかに関係なく、同じイベントが配信されます。回線をまたいで参加（Join Across Lines）機能が実行されると、1 つの会議に結合されるコンサルト コールまたは最後のコールが存在するコントローラの端末上のすべてのアドレスに、CiscoConferenceStartEv/EndEv が提供されます。

CiscoConferenceStartEv では、conferenceControllerAddress が必ずプライマリ コントローラ アドレスになります。アプリケーションで setConferenceController() API を使用してコントローラを設定できるようになりました。アプリケーションでコントローラが設定されていない場合、JTAPI によって会議に適切なコントローラが検出されます。回線をまたいで参加（Join Across Lines）機能が起動されている場合、アプリケーションでコントローラ アドレスを設定することをお勧めします。

オブザーバをコントロール アドレスに追加しないと、アプリケーションでは、CiscoConferenceStartEv の通話中または保留中端末のいずれかの Connection 値が、null 値と認識される可能性があります。このリリースよりも前のリリースでは、アプリケーションが回線をまたいで会議に参加しようとすると、その要求は JTAPI レイヤで失敗していました。このリリースでは、conference() API の実装が強化されたため、最後のコールとコンサルト コールに適した TerminalConnection が検出された後はすべての要求がパス スルーします。JTAPI は、コールに関与する複数のアドレスに共通の端末に基づいて、適切な TerminalConnection を検出します。2 つ以上のコールが参加する必要がある場合、異なるアドレス間での複数の会議もサポートされます。5.1.2 リリースの SIP デバイスではこの機能をサポートしていません。SIP デバイスでこの機能が要求された場合、JTAPI により例外（ILLEGAL_HANDLE）がスローされます。

この機能には、インターフェイスの変更はありません。アプリケーションに提供されるイベントによって、動作は変わります。

下位互換性

この機能が無効になっている場合は、会議の動作は変更されないため、この機能は下位互換性があります。この機能は、デバイスごとに有効または無効に設定できます。デバイスの回線をまたいで参加設定をデフォルトに設定すると、システム全体の CallManager サービス パラメータの Join Across Lines Policy 設定が使用されます。この機能を有効に設定し、アプリケーションが回線をまたいで参加（Join Across Lines）を実行した場合、上記のように、動作に違いがあります。

リリース 5.1 用に作成された JTAPI アプリケーションは、リリース 5.1.2 と同時にリリースされた JTAPI と下位互換性があります。JTAPI クライアントのアップグレードを考慮するのは、新しい機能を使用する場合だけです。

CiscoTermRegistrationFailedEv の新しいエラー コード

このイベントは、TerminalRegistration が何らかの理由で失敗したときに、アプリケーションに送信されます。getErrorCode() インターフェイスの戻り値で、障害の種類が示されます。このイベントを受け取った場合、アプリケーションは Terminal の再登録を試みる必要があります。このバージョンでは、新しい戻り値がこのインターフェイスに追加されました。不明な障害を処理するために、CiscoTermRegistraionFailedEv.UNKNOWN が、このバージョンから導入されました。

下位互換性

この機能は下位互換性があります。

アスタリスク（*）50 の更新

アスタリスク（*）50 機能を使用すると、ユーザは電話機の UI から、コールを元の着信者（CiscoCall.getCalledAddress() メソッドによる戻り値）と、着信者（CiscoCall.getCurrentCalledAddress() メソッドによる戻り値）に転送できます。[iDivert] ソフトキーを押すと、メニューに元の着信者と着信者の名前が表示されます。

ユーザがこの 2 つ名前のうちいずれかを選択すると、コールは選択した着信者のボイス メールボックスに転送されます。従来の即転送機能では、[iDivert] ソフトキーを押すだけで、コールは元の着信者のボイス メールボックスに転送されます。Cisco Unified Communications Manager Administration には、この機能を設定するために、次のサービス パラメータが導入されています。

- iDivert Legacy Behavior : ユーザが [iDivert] ソフトキーを押したときに、電話機が、従来の即転送機能と拡張された *50 即転送機能のどちらの動作をするかを指定します。iDivert legacy サービス パラメータが true に設定されている場合は、従来の即転送の動作が適用され、false に設定されている場合は、従来の即転送機能適用されません。

- Allow QSIG during iDivert : 従来の即転送機能の使用を、QSIG トランクを越えたボイス メッセージングが統合された構成内でも許可するか、Use Legacy iDivert サービス パラメータが true に設定されている場合に限定するかを指定します。
- iDivert User Response timer : Cisco Unified Communications Manager Administration が、iDivert 画面が削除される前に、ユーザからの応答を待つ時間を秒単位で指定します。この時間が経過してもユーザからの操作がない場合、タイマーは期限切れとなり、電話機から画面が削除されます。Use Legacy iDivert サービス パラメータが true に設定されている場合、Cisco Unified Communications Manager Administration はこのパラメータを無視します。

この機能では、JTAPI レイヤに対するインターフェイスの変更はありません。JTAPI アプリケーション側から見た動作の変更点は、コールは、元の着信者または着信者のいずれかのボイス メールに転送される可能性があることを示します。

下位互換性

この機能は下位互換性があります。

コール転送のオーバーライド

この機能では、コール転送のすべての機能をオーバーライドするメカニズムが提供されます。ユーザ（CFA の開始者）が CFA を別のユーザ（CFA ターゲット）に設定した場合、CFA ターゲットが CFA の開始者にコールした場合は、CFA は無視される必要があります。これにより、CFA ターゲットは重要なコールを CFA の開始者に着信させることができます。

この CallManager 機能の動作は、サービス パラメータ（CFADestinationOverride）を使用して設定できます。

例 : Alice は DN 1000 の電話機を所持しています。* Bob は DN 2000 の電話機を所持しています。* Daniel は DN 4000 の電話機を所持しています。* Alice は CFA を 2000 に設定しています。

CFA の動作を次に示します。* Bob が Alice にコールするとします。コールは Alice に着信し、CFA の設定に従った Bob に戻るという動作はしません。* Daniel が Alice にコールするとします。このコールは CFA 設定に従って Bob に転送されます。* Bob はコールに応答し、Alice にそのコールを転送します。Alice が自分への電話を Bob に転送したため、Bob はこの操作を実行できます。この機能では、JTAPI レイヤに対するインターフェイスの変更はありません。ただし、CiscoAddress.setForward() API の起動時には、JTAPI アプリケーションでは異なる動作が確認される場合があります。例のように、CFA ターゲットが CFA の開始者にコールするシナリオでは、機能が有効に設定されている場合、コールは転送されません。

下位互換性

リリース 5.0 用に作成された JTAPI アプリケーションは、リリース 5.1 と下位互換性があります。JTAPI クライアント アップグレード アプリケーションでは、その実行や下位互換性のための JTAPI クライアントのアップグレードは必要ありません。JTAPI クライアントのアップグレードが必要なのは、新しい機能を使用する場合だけです。

パーティションのサポート

Cisco Unified Communications Manager リリース 5.0 よりも前では、JTAPI はパーティションをサポートしていませんでした。JTAPI では、DN が同じでパーティションが異なるアドレスが、同じアドレスと見なされていました。このような場合、アドレスが DN のみによって識別され、パーティション情報によって識別されないため、Address オブジェクトが 1 つしか作成されませんでした。

リリース 5.0 からは、同じ DN を持ちながら異なるパーティションに属するアドレスがサポートされ、それぞれを異なるアドレスとして扱うことが可能になりました。アドレスのパーティション情報は、後述のメソッドを通じてアプリケーションに公開されます。アプリケーションでこのパーティション サポート機能を利用するには、JTAPI インターフェイスを通じて提供される API と、それに応じたアドレス オブジェクトを使用する必要があります。

この機能は下位互換性があります。JTAPI は、アドレス オブジェクトのオープンとアクセスに現在使用されている API をサポートしています。

Cisco Unified Communications Manager リリース 5.0 では、JTAPI がパーティションに対応しており、次の構成がサポートされています。

- パーティションが同じで、デバイスが異なる同一 DN のアドレスは、共用回線として扱われます。
- 同じパーティションと同じデバイス内の同一 DN のアドレスは使用できません。
- パーティションが異なり、同じデバイスにある同一 DN のアドレスは、異なるアドレスとして扱われます。この場合は 2 つのアドレス オブジェクトが作成され、アプリケーションではアドレス オブジェクトの getPartition() API を呼び出すことによってこれら 2 つを区別できます。
- パーティションが異なり、デバイスが異なる同一 DN のアドレスは、異なるアドレスとして扱われます。この場合は 2 つのアドレス オブジェクトが作成され、アプリケーションではアドレス オブジェクトの getPartition() API を呼び出すことによってこれら 2 つを区別できます。

JTAPI でのパーティション サポートに関する変更は、アドレス オブジェクトに限定されており、JTAPI の他の関数やクラスには影響はありません。次のセクションではインターフェイスの変更点について説明します。

CiscoAddress インターフェイス

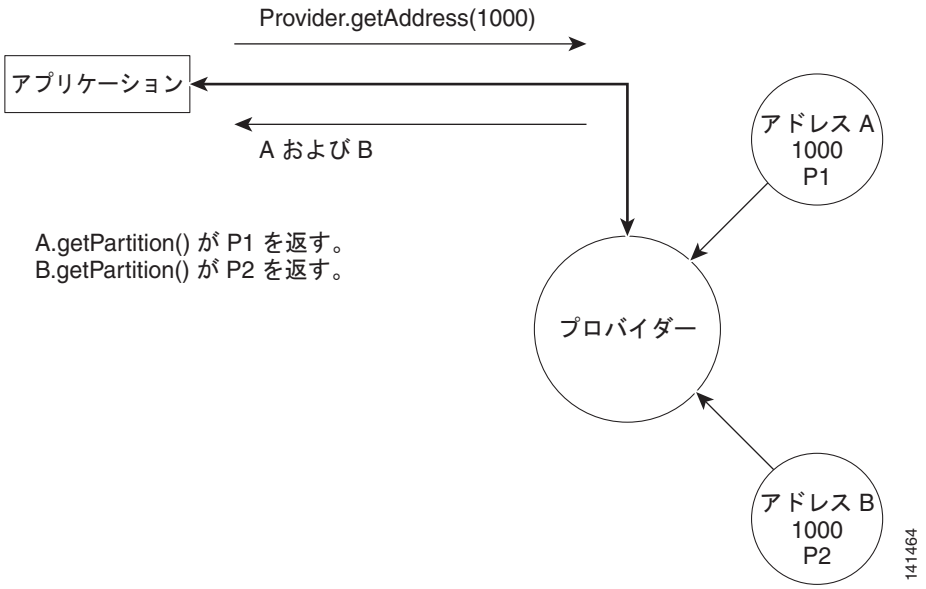
このクラスでは、次のシグニチャを持つ新しいメソッドが提供されています。

string **getPartition ()**

アドレス オブジェクトのパーティション文字列を返します。アプリケーションがパーティション情報を取得するにはこのメソッドを使用する必要があります。JTAPI ではこのパーティション情報を使用して、DN が同じでパーティションが異なるアドレスを区別し、特定のアドレスを開くためのパーティション情報を送信します。

たとえば、プロバイダーのオープンで A(1000, P1) および B(1000, P2) という 2 つのアドレスが返されたとします。A と B はアドレス オブジェクト、1000 はアドレス オブジェクトの DN、P1 と P2 はアドレスが属するパーティションを表します。

図 3-3　プロバイダーのオープンで 2 つのアドレスが返される



ユーザが A.getPartition () を呼び出すと P1 が返され、B.getPartition () を呼び出すと P2 が返されます。

provider.getAddresses() メソッドは、DN が同じでパーティション情報が異なる Address オブジェクトの場合に、複数のアドレスを返します。アプリケーションでこのメソッドを使用することで、DN が同じで異なるパーティションに属する 2 つの Address オブジェクトを区別できます。

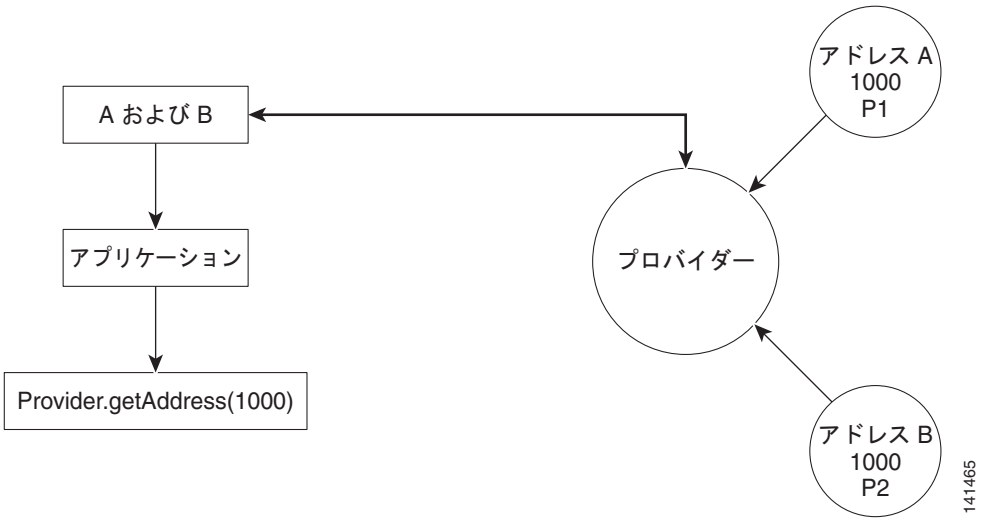
CiscoProvider インターフェイス

CiscoProvider インターフェイスでは次のメソッドが提供されています。

Address[]	**getAddress**(String number) 各パーティションの number に対応する Address オブジェクトの配列を返します。
Address	**getAddress**(String number, String partition) number パラメータと同じ DN を持ち、partition パラメータで指定されているのと同じパーティションに属する Address オブジェクトを返します。

A(1000, P1) および B(1000, P2) という 2 つのアドレスがあるとします。A と B はアドレス オブジェクトで、1000 は 2 つのアドレス オブジェクトの DN を表しており（どちらも同じ値になっています）、P1 および P2 は各アドレスが属するパーティションを示します。アプリケーションが provider.getAddress("1000") をコールすると、A および B という 2 つのアドレス オブジェクトが取得されます。

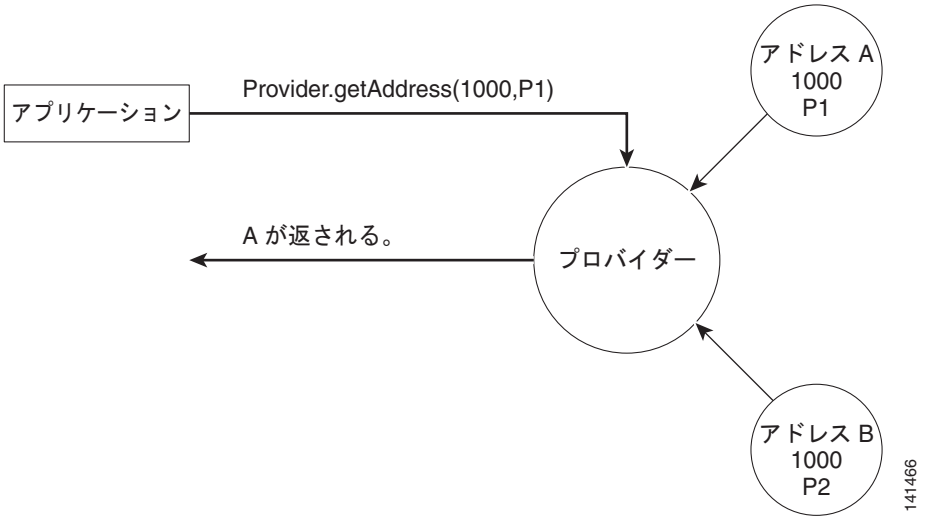
図 3-4　provider.GetAddress() によって 2 つのアドレス オブジェクトが返される



アプリケーションで A.getPartition() を呼び出すと P1 が返され、B.getPartition() を呼び出すと P2 が返され、以降、同様です。アプリケーションでは getPartition メソッドを使用して、2 つのアドレス オブジェクトを区別できます。

アプリケーションで provider.getAddress(1000, P1) を呼び出した場合を考えます。この場合、アプリケーションでは、DN が 1000 でパーティションが P1 のアドレス オブジェクトを探します。この場合は、プロバイダー オブジェクトによって「A」が返されます。

図 3-5　プロバイダーが特定のアドレスとパーティションを呼び出す



CiscoProvCallParkEv イベント

CiscoProvCallParkEv は次のメソッドをこのインターフェイスで提供します。

string `getParkingPartyPartition()`

パークしている通話者のパーティション文字列を返します。

string `getParkedPartyPartition()`

パークされた通話者のパーティション文字列を返します。

string　`getParkPartyPartition()`

パーク DN のパーティション文字列を返します。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。パーティションのサポートに関するメッセージ シーケンスを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

ヘアピン サポート

ヘアピン コールは、1 つのクラスタを離れゲートウェイを経由して他のデバイスに発信されたコールが、同じクラスタのデバイスに戻ってきたときに発生します。どちらのコールも同じクラスタ内にありますが、クラスタに戻ってきたコールの GCID とコールを発信した元の GCID は異なります。以前のリリースでは、両方が JTAPI で制御されている場合は、2 つの Connection（CiscoAddress.Internal の Connection と CiscoAddress.External の Connection）がありました。

JTAPI では、アプリケーションがヘアピン コールの両端を監視する場合に、ヘアピン コールがサポートされます。以前はアドレスが DN だけで表現されていたため、ヘアピン コールの一端しか監視できませんでした。

現在のリリースでは、DN が同じ 2 つのアドレスが存在し、一方が同じクラスタ内にあり、他方がゲートウェイを経由する場合に、JTAPI により外部 DN に対して別々のアドレス オブジェクトが作成され、アドレスのタイプごとに Connection が 1 つだけ返されるようになっています。以前のリリースではアドレスが常に DN で表されており、アプリケーションでアドレスの Connection を取得すると 2 つの Connection を取得していましたが、このプロセスによりヘアピンの問題が回避されます。

これらの問題を個別に解決すると以前のリリースとの互換性に問題が生じる恐れがあるため、このリリースにはこれらの問題に対処するための包括的なソリューションが組み込まれています。監視対象のローカル通話者と同じ DN を持つ外部通話者に関連するコールについては、適切にサポートされるようになりました。ただし、この機能に対応した新しいインターフェイスは追加されていません。

下位互換性

この機能には下位互換性はありません。

QoS のサポート

このリリースでは QoS のサポートが拡張され、アプリケーションと CTIManager 間の接続の双方向で QoS（DSCP マーキング）が可能になりました。以前のリリースでは、CTIManager からアプリケーションへの一方向の QoS のみがサポートされていました。

このリンクの双方向の DSCP（QoS）値は、CTIManager のサービス パラメータ「DSCP IP CTIManager to Application」の値によって設定されます。デフォルト値は CS3（優先順位 3）DSCP（011000）です。

「DSCP value for Audio calls」サービス パラメータは音声コールに対して推奨される QoS 値です。この値は JTAPI アプリケーションに公開されます。

JTAPI の QoS が Windows プラットフォームで機能するためには、次の設定手順をクライアント マシンで実行する必要があります。

Windows 2000 を実行している場合は、次の手順に従います。

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　次のキーに移動します：
Local Machine¥System¥CurrentControlSet¥Services¥Tcpip¥Parameters¥ 上の HKEY_LOCAL_MACHINE

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　[値の名前] ボックスに、**DisableUserTOSSetting** と入力します。

ステップ 5　[データ型] リストで、[REG_DWORD] をクリックし、[OK] をクリックします。

ステップ 6　[データ] ボックスに、値 **0** を入力し、[OK] をクリックします。

ステップ 7　レジストリ エディタを終了し、コンピュータを再起動します。

Windows XP または Windows Server 2003 を実行している場合は、次の手順に従います。

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　次のキーに移動します：
Local Machine¥System¥CurrentControlSet¥Services¥Tcpip¥Parameters¥ 上の HKEY_LOCAL_MACHINE

ステップ 3　[編集] メニューで、[新規] をポイントし、[DWORD 値] をクリックします。

ステップ 4　エントリ名として **DisableUserTOSSetting** と入力し、**Enter** キーを押します。
このエントリを追加すると、値が 0 に設定されます。値は変更しないでください。

ステップ 5　レジストリ エディタを終了し、コンピュータを再起動します。

レジストリ エディタを使用して Internet Protocol Type of Service の各ビットを設定する方法の詳細については、Microsoft のテクニカル サポート Web サイトの「Setsockopt is unable to mark the Internet Protocol type of service bits in Internet Protocol packet header」というトピックを参照してください。

次の JTAPI インターフェイスでは QoS をサポートしています。

プロバイダー インターフェイス

int	**`getAppDSCPValue()`** 「DSCP IP for CTI applications」サービス パラメータを返します。この値によって、JTAPI が CTI へのリンクに対して設定する DSCP 値が指定されます。アプリケーションは、ProviderInServiceEvent を受け取るたびにこの API を使用してプロバイダー オブジェクトを照会すれば、この値を取得できます。
private int	`precedenceValue = 0x00` CTI によって提供された DSCP 値を保存します。

これらのインターフェイスの詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。QoS のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

トランスポート レイヤ セキュリティ（TLS）

この機能を使用すると、JTAPI アプリケーションと CTIManager の間でセキュアな接続を通じて通信できます。CTIManager は JTAPI からの接続を受け入れるために TLS リスナー ソケットを実行します。TLS 接続を確立するには、クライアント認証のためにサーバが使用するクライアント証明書と、サーバ認証のためにクライアントが使用するサーバ証明書が必要です。

Cisco Unified Communications Manager 環境では、サーバ証明書は TFTP サーバ上に CTL 形式で存在し、JTAPI はこの証明書をダウンロードします。CTL の最初のダウンロードは信頼され、検証なしで行われるため、このダウンロードはセキュアな環境で実行することを強くお勧めします。CTL は、CTL ファイル内に存在する 2 つの System Administrator Security Tokens（SAST）の一方によって署名され、その後の CTL のダウンロードは古い CTL ファイルの SAST を使用して検証されます。

JTAPI は CAPF プロトコルを使用して CAPF に接続し、クライアント証明書（LSC）を取得します。これらの証明書は、CTL 内に存在する発行元の証明書を使用して認証できます。

CTI は、クライアント証明書ごとに作成されるプロバイダー接続の数を追跡します。アプリケーションでは、クライアント証明書を使用してプロバイダーを 1 つだけ作成できます。プロバイダーのインスタンスを複数作成した場合は、両方のプロバイダーが CTI から切断され、アウト オブ サービスになります。JTAPI では、元のプロバイダーがイン サービスになるように CTI への接続を再試行しますが、プロバイダーの両方のインスタンスが引き続き存在する場合は、一定回数の再試行の後、プロバイダーが恒久的にシャットダウンされ、クライアント証明書は信頼のおけないものとしてマークされます。それ以降は、このクライアント証明書を使用してプロバイダーを作成しようとしても失敗します。アプリケーションでは、管理者に連絡して新しいインスタンス ID を設定し、新しいクライアント証明書をダウンロードして操作を再開する必要があります。

（注）　クライアント証明書にはそれぞれ、Cisco Unified Communications Manager データベースで設定された一意のインスタンス ID が関連付けられています。アプリケーションでは、providerString でオプションのパラメータとしてインスタンス ID を提供することで、CiscoProvider の作成時に一意の証明書を使用できます。

TLS を使用して複数のアプリケーション インスタンスを実行するには、アプリケーション ユーザが複数のインスタンス ID で Cisco Unified Communications Manager データベースに登録されていることを確認します。これらの一意のインスタンス ID は、各インスタンスに固有のクライアント証明書を取得するためにアプリケーションによって使用されます。

JTAPI Preferences アプリケーションには、セキュリティ パラメータの設定とサーバ/クライアント証明書の更新を行うための GUI が用意されています。アプリケーション ユーザは、アプリケーション サーバの証明書をダウンロードしてインストールするために、JTAPI Preferences で TFTP サーバの IP アドレス、CAPF サーバの IP アドレス、ユーザ名、インスタンス ID、AuthorizationString の各パラメータを設定する必要があります。

クライアント レイヤ オブジェクトに関する新しいインターフェイスが JTAPI クライアント アプリケーション用に追加されました。たとえば、CTIClientProperties クラスに関する JTAPI クライアント インターフェイスが用意されています。

この機能は以前のリリースと下位互換性があり、JTAPI アプリケーションは引き続きセキュアでないソケット接続でも CTIManager に接続できます。ただし、このオプションを使用できるリリースは限定されているため、将来のリリースでは削除される可能性があります。セキュアな接続を使用するようにアプリケーションを修正することをお勧めします。

次のセクションでは、TLS をサポートするための JTAPI でのインターフェイスの変更点について説明します。

CiscoJtapiPeer.getProvider()

```
public javax.telephony.Provider getProvider(java.lang.String providerString)
throws
javax.telephony.ProviderUnavailableException
```

この変更されたインターフェイスは、新しいオプションのパラメータである InstanceID を受け取ります。必要なサービス名を含む文字列引数を渡すと、Provider オブジェクトのインスタンスを返します。

次の形式で、オプションの引数をこの文字列に提供することもできます。

```
< service name > ; arg1 = val1; arg2 = val2; ...
```

< service name > は必須で、オプションの引数＝値の各ペアはセミコロンで区切って指定します。これらの引数のキーは、次の 2 つの標準定義キーを除いて、実装に固有のものになります。

- login：ログイン ユーザ名を Provider に提供します。
- passwd：パスワードを Provider に提供します。

CiscoJtapiPeer の providerString では新しいオプションの引数が使用できます。

- InstanceID：アプリケーション インスタンスのインスタンス ID を提供します。

InstanceID は、アプリケーションの複数のインスタンスが同じクライアント マシンから TLS 接続を通じて Provider（CTIManager）に接続する場合に必要になります。アプリケーションの各インスタンスでは、TLS 接続を確立するために、それぞれ一意の X.509 証明書が必要になります。JTAPI で同じユーザ名またはインスタンス ID を使用して複数の接続をオープンしようとした場合は、CTIManager によって TLS 接続が拒否されます。インスタンス ID を提供しない場合は、JTAPI によって USER のインスタンスが 1 つランダムに選択されます。その場合、選択されたインスタンスに対応する接続がすでに存在していると、接続が失敗することがあります。

引数が null の場合、このメソッドは、JtapiPeer オブジェクトによって決められた何らかのデフォルトのプロバイダーを返します。返されるプロバイダーの状態は Provider.OUT_OF_SERVICE になります。

事後条件：

```
this.getProvider().getState() = Provider.OUT_OF_SERVICE
```

指定元：インターフェイス `javax.telephony.JtapiPeer` の `getProvider`。

パラメータ：`providerString`：必要なサービスの名前と、オプションの引数。

戻り値：Provider オブジェクトのインスタンス。

例外：`javax.telephony.ProviderUnavailableException`：指定された文字列に対応するプロバイダーが使用できないことを示します。

CiscoJtapiProperties

JTAPI では、セキュリティ オプションを有効または無効にしてセキュアな TLS ソケット接続の確立に必要なクライアント/サーバ証明書をインストールするためのインターフェイスが CiscoJtapiProperties で提供されます。

```
com.cisco.jtapi.extensions
Interface CiscoJtapiProperties
```

---

getSecurityPropertyForInstance

```
public java.util.Hashtable getSecurityPropertyForInstance()
```

このインターフェイスは、User/InstanceID に対するすべてのパラメータが設定されたハッシュ テーブルを返します。ハッシュ テーブルは次の「キー」と「値」のペアで設定されます。

表 3-3

キー	値
「user」	userName
文字列「instanceID」	InstanceID
文字列「AuthCode」	authCode
文字列「CAPF」	capfServer の IP アドレス
文字列「CAPFPort」	capfServer の IP アドレス / ポート
文字列「TFTP」	tftpServer の IP アドレス
文字列「TFTPPort」	tftpServer の IP アドレス / ポート
文字列「CertPath」	証明書パス
文字列「securityOption」	セキュリティ オプションを表すブール値（有効なら true、無効なら false）。
文字列「certificateStatus」	証明書のステータスを表すブール値（更新済みなら true、未更新なら false）。

戻り値：上記の形式による、最初のユーザおよびインスタンスのハッシュ テーブル。

getSecurityPropertyForInstance

```
public java.util.Hashtable getSecurityPropertyForInstance
(java.lang.String user, java.lang.String instanceID)
```

このインターフェイスは、User/InstanceID に対するすべてのパラメータが設定されたハッシュ テーブルを返します。ハッシュ テーブルは次の「キー」と「値」のペアで設定されます。

表 3-4

キー	値
「user」	userName
文字列「instanceID」	InstanceID
文字列「AuthCode」	authCode
文字列「CAPF」	capfServer の IP アドレス
文字列「CAPFPort」	capfServer の IP アドレス / ポート
文字列「TFTP」	tftpServer の IP アドレス
文字列「TFTPPort」	tftpServer の IP アドレス / ポート
文字列「CertPath」	証明書パス
文字列「securityOption」	セキュリティ オプションを表すブール値（有効なら true、無効なら false）。
文字列「certificateStatus」	証明書のステータスを表すブール値（更新済みなら true、未更新なら false）。

パラメータ：

`user`：セキュリティ パラメータを取得する UserName。

`instanceID`：セキュリティ パラメータを取得する InstanceID。

戻り値：上記の形式のハッシュ テーブル。

---

setSecurityPropertyForInstance

```
public void setSecurityPropertyForInstance(java.lang.String user,
                      java.lang.String instanceID,
                      java.lang.String authCode,
                      java.lang.String tftp,
                      java.lang.String tftpPort,
                      java.lang.String capf,
                      java.lang.String capfPort,
                      java.lang.String certPath,
                      boolean securityOption)
```

このインターフェイスを使用して、次のパラメータのセキュリティ プロパティを設定できます。

パラメータ：

`user`：セキュリティ パラメータを更新するユーザ名。

`instanceID`：セキュリティ パラメータを更新するインスタンス ID。

`authCode`：認証文字列。

`capf`：CAPF サーバの IP アドレス。

`capfPort`：CAPF サーバが稼動中の IP アドレスのポート番号。これは CallManager のサービス パラメータで定義されます。値が null の場合、デフォルト値は 3804 です。

`tftp`：TFTP サーバの IP アドレス。

`tftpPort`：TFTP サーバが稼動している IP アドレス / ポート番号。通常、Cisco Unified Communications Manager TFTP サーバはポート 69 上で実行されます。値が null の場合、デフォルト値は 69 です。

`certPath`：証明書をインストールする必要があるパス。

---

updateCertificate

```
public void updateCertificate(java.lang.String user,
               java.lang.String instanceID,
               java.lang.String authcode,
               java.lang.String ccmTFTPAddress,
               java.lang.String ccmTFTPPort,
               java.lang.String ccmCAPFAddress,
               java.lang.String ccmCAPFPort,
               java.lang.String certificatePath)
```

このインターフェイスは、Cisco Unified Communications Manager Certificate Authority Proxy Function（CAPF）サーバに接続することによって、USER インスタンスの X.509 クライアント証明書を証明書ストアにインストールします。また、Cisco Unified Communications Manager TFTP サーバから Certificate Trust List（CTL; 証明書信頼リスト）をダウンロードします。

ユーザ クレデンシャルが無効な場合、このメソッドは PrivilegeViolationException をスローします。TFTP サーバまたは CAPF サーバのアドレスが不正な場合、このメソッドは InvalidArgumentException をスローします。アプリケーションのすべてのインスタンスには、それぞれ固有のクライアント証明書が必要です。Cisco Unified Communications Manager データベースで複数の instanceID が設定されている場合、アプリケーションはこのインターフェイスを複数回実行してすべてのインスタンスのクライアント証明書をインストールすることができます。

事前条件：このインターフェイスを実行する際は、アプリケーションが Cisco Unified Communications Manager CAPF サーバおよび TFTP サーバとネットワーク接続されている必要があります。

事後条件：これによって、クライアント証明書とサーバ証明書が JTAPI アプリケーションのマシンにインストールされます。

パラメータ：

`user`：Cisco Unified Communications Manager データベースで設定された CTI アプリケーションのユーザの名前。

`instanceID`：Cisco Unified Communications Manager データベースで設定されたアプリケーションのインスタンス ID。アプリケーションのすべてのインスタンスには、一意の ID が必要になります。

`authCode`：Cisco Unified Communications Manager データベースで設定された認証文字列。authCode は証明書を取得するために一度だけ使用できます。

`ccmTFTPAddress`：Cisco Unified Communications Manager TFTP サーバの IP アドレス。

`ccmTFTPPort`：Cisco Unified Communications Manager TFTP サーバが稼動している IP アドレス / ポート番号。通常、Cisco Unified Communications Manager TFTP サーバはポート 69 上で実行されます。null の場合、デフォルト値は 69 です。

`ccmCAPFAddress`：Cisco Unified Communications Manager CAPF サーバの IP アドレス。

`ccmCAPFPort`：Cisco Unified Communications Manager CAPF サーバが稼動中のポート番号。これは Cisco Unified Communications Manager のサービス パラメータで定義されます。値が null の場合、デフォルト値は 3804 です。

`certificatePath`：証明書をインストールする必要があるディレクトリ パス。

例外：

`InvalidArgumentException`：指定した TFTP サーバ アドレスまたは CAPF サーバ アドレスが無効な場合、この例外がスローされます。

`PrivilegeViolationException`：指定した user、instanceID、または authCode が無効な場合、この例外がスローされます。

IsCertificateUpdated

```
public boolean IsCertificateUpdated
    (java.lang.String user, java.lang.String instanceID)
```

このインターフェイスは、指定した user/instanceID のクライアントおよびサーバの証明書が更新されているかどうかに関する情報を提供します。

パラメータ：

`user`：Cisco Unified Communications Manager Administration に定義されたユーザ名。

`instanceID`：指定したユーザ名のインスタンス ID。

戻り値：証明書が更新済みの場合は true、未更新の場合は false。

---

updateServerCertificate

```
public void updateServerCertificate(java.lang.String ccmTFTPAddress,
                  java.lang.String ccmTFTPPort,
                  java.lang.String ccmCAPFAddress,
```

```
                    java.lang.String ccmCAPFPort,
                    java.lang.String certificatePath)
```

このインターフェイスは、証明書パスに指定された X.509 サーバ証明書をインストールします。TFTP サーバのアドレスが不正な場合、このメソッドは InvalidArgumentException をスローします。自動アップデートを行うアプリケーションは、Cisco Unified Communications Manager との HTTPS 接続を呼び出す前に、このインターフェイスを使用してサーバ証明書を更新する必要があります。

事前条件：このインターフェイスを実行する際は、アプリケーションが TFTP サーバとネットワーク接続されている必要があります。

事後条件：これによって、サーバ証明書が JTAPI アプリケーションのマシンにインストールされます。

パラメータ：

`ccmTFTPAddress`：Cisco CallManager TFTP サーバの IP アドレス。

`ccmTFTPPort`：Cisco Unified Communications Manager TFTP サーバが稼動中のポート番号。null の場合、デフォルト値は 69 です。

`certificatePath`：証明書をインストールするディレクトリ パス。

`ccmCAPFAddress`：Cisco Unified Communications Manager CAPF サーバの IP アドレス。

`ccmCAPFPort`：Cisco Unified Communications Manager CAPF サーバが稼動中のポート番号。値が null の場合、デフォルト値は 3804 です。

例外：

`InvalidArgumentException`：TFTP サーバのアドレスが無効な場合。

JTAPI Preferences で提供されるインターフェイス

[JTAPI Preferences] ダイアログボックスには [セキュリティ] タブが提供されています。アプリケーション ユーザはこのタブから、ユーザ名、インスタンス ID、authCode、TFTP IP アドレス、TFTP ポート、CAPF IP サーバ アドレス、CAPF サーバ ポート、証明書パスをそれぞれ設定でき、セキュアな接続を有効にすることができます。

- [CAPF server port] 番号のデフォルトは 3804 です。

  この値は Cisco Unified Communications Manager Administration サービス パラメータ ウィンドウで設定できます。JTAPI Preferences から入力する CAPF サーバ ポート値は、Cisco Unified Communications Manager Administration ページで設定した値と同じである必要があります。

- [TFTP server port] 番号のデフォルトは 69 です。

  システム管理者からの指示がない限り、この値は変更しないでください。

- [Certificate Path] は、アプリケーションがサーバ証明書とクライアント証明書をインストールする場所です。

  このフィールドが空の場合、証明書は JTAPI.jar のクラスパスにインストールされます。

- [Certificate Update Status] には、証明書の更新状態に関する情報が表示されます。

- Cisco Unified Communications Manager へのセキュアな TLS 接続を有効にするには、[Enable Secure Connection] を選択する必要があります。

  [Enable Secure Connection] が選択されていない場合は、証明書が更新またはインストールされていても、JTAPI では CTI へのセキュアではない接続が行われます。

- [Enable Security Tracing] チェックボックスで、証明書のインストール操作のトレースを有効または無効にできます。

トレースを有効にした場合は、ドロップダウン メニューから [Error]、[Debug]、[Detailed] の 3 つの異なるレベルを選択できます。

JTAPI Preference UI を使用して、ユーザ名とインスタンス ID の 1 つ以上のペアに対してセキュリティ プロファイルを設定できます。ユーザ名とインスタンス ID のペアに対してすでにセキュリティ プロファイルが設定されている場合、アプリケーション ユーザがこのウィンドウに再度アクセスし、ユーザ名とインスタンス ID を入力して他の編集ボックスをクリックすると、セキュリティ プロファイルが自動的に編集ボックスに入力されます。

JTAPI Preference UI の [Trace Levels] タブの名前は [JTAPI トレース] に変更されました。この変更により、[JTAPI トレース] タブで変更できるトレース設定が JTAPI 層のトレース設定のみであることが明確になりました。セキュリティ証明書のインストールに対するトレースについては、[セキュリティ] タブで有効にする必要があります。

SIP フォンのサポート

このリリースの Cisco Unified Communications Manager では、SIP を実行する電話を登録して SCCP を実行する電話と相互運用することができます。以降のセクションでは、SIP を実行する電話をサポートするために導入された新しいインターフェイスに加えて、SCCP をサポートする電話と比較した場合の動作の制限および相違点について説明します。既存のすべての機能が SIP を実行する電話でサポートされるわけではありませんが、JTAPI のイベントとインターフェイスの点で、SIP を実行する電話の全般的な動作は SCCP を実行する電話とほぼ同じです。

JTAPI アプリケーションで制御できるのは SIP を実行する Cisco Unified IP Phone 7900 シリーズ（Cisco Unified IP 7970 フォンなど）のみです。Cisco Unified IP 7960 や 7940 などの SIP プロトコルを実行する電話機はアプリケーションの制御リストに含めないでください。また、サード パーティ製の SIP を実行する電話も JTAPI アプリケーションでは制御できないので、制御リストには含めないでください。

以前のリリースでは、JTAPI は SIP を実行する電話で初期的なフィーチャ セットをサポートしていました。このリリースでは、SIP を実行する電話で次の機能のサポートが追加されました。

- SIP を実行する電話のパーク
- SIP を実行する電話のパーク解除

ヒント　SIP を実行する電話と SCCP フォンでは、コンサルト コールのイベントの順序が異なります。次のような例が考えられます。

a. 端末 A が共用回線 B/B' にコールを発信します。

b. 共用回線が端末 C にコンサルト コールを発信します。

共用回線が SIP デバイスの場合、次のコール イベントが発生します。

- B（アクティブ）は OnHold -> Select -> NewCall の順に受信します。
- B'（使用中のリモート）は Select -> NewCall -> OnHold の順に受信します。

ただし、共用回線が SCCP デバイスの場合、コール イベントは両方の端末で Select -> OnHold -> NewCall の順になります。

アプリケーションがモニタリングだけを行っている場合、call.getConsultingTerminalConnection() によって null が返される可能性があります。

JTAPI は SIP を実行する電話で次の機能をサポートしています。

- Call.connect、offhook

- answer、disconnect、drop、hold、unhold
- consult、transfer、conference、redirect
- playdtmf、deviceData

JTAPI は SIP を実行する電話で次のイベントをサポートしています。

- CiscoTermDeviceStateEv、RTP イベント、inService および OutOfService
- MediaTermConnDtmfEv（アウトバンドのみ）、転送開始および終了イベント、会議の開始および終了イベント、CiscoToneChangedEv および CiscoTermConnPrivacyChangedEv

SIP を実行する電話の動作は、次の点で SCCP を実行する電話と異なります。

- コール拒否：SIP を実行する電話に対してコールが発信されたとき、電話はそのコールを拒否することを選択できます。この場合、アプリケーションはその SIP 端末のアドレスに関する CallActive, ConnCreatedEv を受信した後に ConnDisconnectedEv を受信します。これは RP がコールを拒否した場合とほぼ同じです。
- メディアのないコンサルト コールに SIP フォンが含まれている場合は、接続後 1.5 秒以内に転送する必要があります。
- SIP を実行する電話では常に一括ダイヤルが使用されます。これはユーザが最初にオフフックしてから番号をダイヤルする場合も同じです。SIP フォンはすべての番号を収集してから、その番号を Cisco Unified Communications Manager に送信します。そのため、CallCtlConnDialingEv は常に、いずれかの設定済みダイヤル パターンに一致する桁数の番号が電話機で押された後に配信されます。
- MediaTermConnDtmfEv を受信するには、すべてのデバイスに対して「アウトバンド DTMF」を設定する必要があります。

CTI ポート、ルート ポイント、および SCCP を実行する電話に関するイベントは変更されていません。

トランスポートとして UDP を使用して SIP を実行する Cisco Unified IP Phone 7900 シリーズ モデルと Cisco Unified Communications Manager の間の接続が失われると、JTAPI アプリケーションはその電話機に対して定義されている端末とアドレスに関するイベント CiscoTermOutOfServiceEv と CiscoAddrOutOfServiceEv を受信します。UDP ではその性質上、接続切れが検出されるまでに時間差が生じるため、アプリケーションにアウト オブ サービス イベントが通知された後でも SIP を実行する Cisco Unified IP Phone 7900 シリーズは登録されているように見える場合があります。

Cisco Unified IP Phone 7960、7940、および SIP を実行する Cisco Unified IP Phone 7900 シリーズ以外が制御リストに含まれている場合、オブザーバ（オブザーバとコール オブザーバの両方）がアドレスまたは端末に追加されて CiscoTermRestrictedEv がプロバイダー オブザーバに配信されると、例外がスローされます。これらのイベントの原因は CiscoRestrictedEv.CAUSE_UNSUPPORTED_PROTOCOL です。

端末が SCCP を実行する電話か SIP を実行する電話かを示す新しいインターフェイス getProtocol() が CiscoTerminal で公開されました。getProtocol() が返す値は CiscoTerminalProtocol で定義されています。

CiscoCall に定義されている次の新しいインターフェイスにより、アプリケーションは外部の SIP エンティティの URL 情報を取得できます。

Public interface CiscoCall

CiscoPartyInfo	`getLastRedirectingPartyInfo()`
CiscoPartyInfo	`getCurrentCallingPartyInfo()`

CiscoPartyInfo	**getCurrentCalledPartyInfo()**
CiscoPartyInfo	**getCalledPartyInfo()**

Public interface CiscoPartyInfo

CiscoUrlInfo	**getUrlInfo()**
Address	**getAddress()**
string	**getDisplayName()**
string	**getUnicodeDisplayName()**
boolean	**getAddressPI()**
boolean	**getDisplayNamePI()**
boolean	**getlocale()**

Public interface CiscoUrlInfo

int	**getUrlType()** Final int URL_TYPE_TEL Final int URL_TYPE_SIP Final int URL_TYPE_UNKNOWN
string	**getHost()**
string	**getUser()**
int	**getPort()**
int	**getTransportType()** Final int TRANSPORT_TYPE_UDP Final int TRANSPORT_TYPE_TCP

Public interface CiscoTerminal

int	**getProtocol ()**

CiscoTerminalProtocol

static int	PROTOCOL_NONE プロトコルのタイプが認識不能かまたは不明であることを示します。
static int	PROTOCOL_SCCP デバイスが SCCP を使用して Cisco Unified Communications Manager と通信を行っていることを示します。
static int	PROTOCOL_SIP デバイスが SIP を使用して Cisco Unified Communications Manager と通信を行っていることを示します。

Secure Real-Time Protocol 鍵情報

この機能は、暗号化されたメディア セッションの Secure Real-Time Protocol（SRTP）鍵情報を Cisco Unified Communications Manager をベースとするエンタープライズ システム内の認証済みエンド ポイント間で配信するために必要なメカニズムを提供します。この鍵情報を受け取るには、Cisco Unified Communications Manager の [管理者] ウィンドウで TLS Enabled フラグと SRTP Enabled フラグを設定し、JTAPI と CTIManager の間で TLS リンクを確立する必要があります。

鍵情報は CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv で公開されます。アプリケーションでこれらのイベントを受信するには、CiscoTermEvFilter の rtpKeyEvenabled を有効にする必要があります。デフォルトでは、下位互換性を維持するためにフィルタは無効になっています。フィルタを有効にすると、アプリケーションは常に CiscoRTPInputKeyEv と CiscoRTPOutputKeyEv を受信します。これらのイベントのセキュリティ インジケータは、メディアが暗号化されているかどうか、および鍵が使用可能かどうかを示します。

CiscoRTPInputKeyEv には入力ストリームの鍵情報が含まれ、CiscoRTPOutputKeyEv には出力ストリームの鍵情報が含まれます。この鍵情報は、パケットの復号化してモニタリングを開始したり、メディアを録音する目的に使用できます。この鍵情報は改ざん可能な形で保存してはならず、不要になったときは鍵のエントリをゼロに戻すかクリアする必要があります。

この鍵情報には次の項目が含まれます。

- Key Length
- Master Key
- Salt Length
- Master Salt
- AlgorithmID
- isMKIPresent
- Key Derivation Rate

この機能拡張では、CTIPort と RoutePoint におけるセキュアなメディア終端もサポートされています。これを行うには、サポートされている暗号化アルゴリズムを CTIPort とルート ポイントの登録要求で渡します。TLS リンクが存在せず SRTP Enabled フラグが設定されていない場合は、アプリケーションにエラーが返されます。メディアが暗号化されるかどうかは、相手側がセキュア メディア対応かどうか、およびアルゴリズムのネゴシエーションが成功するかどうかによって決まります。

コール中のモニタリングに関しては、2 つのエンド ポイント間でコールが確立された後にアプリケーションが起動された場合、アプリケーションは Terminal.createSnapshot() とスナップショット イベント CiscoTermSnapshotEv を照会する必要があります。CiscoTermSnapshotCompletedEv が送信され、エンド ポイント間の現在のメディアがセキュアかどうかが示されます。アプリケーションは CiscoMediaCallSecurityIndicator を照会してコールのセキュリティ インジケータを取得できますが、この場合はイベントに鍵情報が含まれません。端末のどの回線にもコールが存在しない場合、アプリケーションは CiscoTermSnapshotCompletedEv のみを取得します。下位互換性を維持するため、これらのイベントはアプリケーションで CiscoTermEvFilter の snapShotRTPEnabled フィルタが有効にされた場合にのみ生成されます。

1 つのコールに関連する RTP イベントを相互に関連付けることができるように、すべての RTP イベントに CiscoRTPHandle が追加されます。下位互換性を維持するため、セキュアなメディアが存在しない場合、新しいイベントは生成されません。

SRTP の詳細については、SourceForge.net の『Secure RTP Library API Documentation』（David McGrew 著）を参照してください。

次のセクションでは、SRTP 鍵情報に関するインターフェイスの変更点について説明します。

Public Interface CiscoMediaEncryptionKeyInfo

int　**getAlgorithmID()**

このメソッドは現在のストリームのメディア暗号化アルゴリズムを返します。

int　**getIsMKIPresent()**

MKI が存在するかどうかを示す MKI インジケータ。MKI は鍵管理によって定義、通知、および使用されます。

int　**getKeyLength ()**

このメソッドはマスター鍵の長さを返します。

byte[]　**getKey()**

このメソッドはストリームのマスター鍵を返します。

int　**getSaltLength ()**

このメソッドはソルトの長さを返します。

byte[]　**getSalt()**

このメソッドはストリームのソルト鍵を返します。

int　**keyDerivationRate()**

このセッションの SRTP 鍵逸脱率を示します。

CiscoMediaSecurityIndicator

static int　MEDIA_ENCRYPTED_KEYS_AVAILABLE

終端されたメディアはセキュリティ保護されており、鍵を参照できます。

static int　MEDIA_ENCRYPTED_KEYS_UNAVAILABLE

メディアはセキュリティを確保したモードで終端されていますが、SRTP が Cisco Unified Communications Manager Administration User ウィンドウで有効にされていないため、鍵は参照できません。このアプリケーションで、TLS が存在しないか IPSec が設定されていない可能性があります。

static int　MEDIA_ENCRYPTED_USER_NOT_AUTHORIZED

メディアはセキュリティを確保したモードで終端されていますが、ユーザが鍵を取得する権限を持っていないため、鍵は参照できません。

static int　MEDIA_NOT_ENCRYPTED

メディアのこのコールは暗号化されていません。

CiscoRTPInputKeyEv

CiscoMediaEncryptionKeyInfo	**getCiscoMediaEncryptionKeyInfo ()** プロバイダーが TLS リンクで開かれており、Cisco Unified Communications Manager User Administration でアプリケーションの SRTP を有効にするオプションが設定されている場合だけ、CiscoMediaEncryptionKeyInfo を返します。そうでない場合は、null を返します。
int	**getCiscoMediaSecurityIndicator()** メディア セキュリティ インジケータを返します。次に示す CiscoMediaSecurityIndicator の定数のうち 1 つを返します。 MEDIA_ENCRYPTED_KEYS_AVAILABLE MEDIA_ENCRYPT_USER_NOT_AUTHORIZED MEDIA_ENCRYPTED_KEYS_UNAVAILABLE MEDIA_NOT_ENCRYPTED
CiscoCallID	getCallID () このイベントの送信時に CiscoCall が存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。
CiscoRTPHandle	getCiscoRTPHandle () CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall(CiscoRTPHandle) を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoRTPOutputKeyEv

CiscoMediaEncryptionKeyInfo	**getCiscoMediaEncryptionKeyInfo ()** プロバイダーが TLS リンクで開かれており、Cisco Unified Communications Manager User Administration でアプリケーションの SRTP を有効にするオプションが設定されている場合にだけ、CiscoMediaEncryptionKeyInfo を返します。そうでない場合は null を返します。
int	**getCiscoMediaSecurityIndicator()** メディア セキュリティ インジケータを返します。次に示す CiscoMediaSecurityIndicator の定数のうち 1 つを返します。 MEDIA_ENCRYPTED_KEYS_AVAILABLE MEDIA_ENCRYPT_USER_NOT_AUTHORIZED MEDIA_ENCRYPTED_KEYS_UNAVAILABLE MEDIA_NOT_ENCRYPTED

CiscoCallID　`getCallID ()`

このイベントの送信時に CiscoCall が存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。

CiscoRTPHandle　`getCiscoRTPHandle ()`

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall（CiscoRTPHandle）を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoTermSnapshotEv

CiscoMediaCallMediaSecurityIndicator[]　**`getMediaCallSecurityIndicator ()`**

このデバイス上の各アクティブ コールにおけるメディアのセキュリティ状態を返します。

CiscoTermSnapshotCompletedEv

このイベントにはメソッドがありません。

CiscoMediaCallSecurityIndicator

int　**`getCiscoMediaSecurityIndicator`**`()`

メディア セキュリティ インジケータを返します。次に示す CiscoMediaSecurityIndicator の定数のうち 1 つを返します。

```
MEDIA_ENCRYPTED_KEYS_AVAILABLE
MEDIA_ENCRYPT_USER_NOT_AUTHORIZED
MEDIA_ENCRYPTED_KEYS_UNAVAILABLE
MEDIA_NOT_ENCRYPTED
```

CiscoCallID　`getCallID ()`

このイベントの送信時に CiscoCall が存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。

CiscoRTPHandle　`getCiscoRTPHandle ()`

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall(CiscoRTPHandle) を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoRTPInputStartedEv

CiscoRTPHandle　getCiscoRTPHandle ()

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall（CiscoRTPHandle）を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoRTPInputStoppedEv

CiscoRTPHandle　getCiscoRTPHandle ()

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall(CiscoRTPHandle) を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoRTPOutputStartedEv

CiscoRTPHandle　getCiscoRTPHandle ()

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall（CiscoRTPHandle）を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoRTPOutputStoppedEv

CiscoRTPHandle　getCiscoRTPHandle ()

CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall(CiscoRTPHandle) を使用してコール参照を取得できます。コール オブザーバが存在しない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall（CiscoRTPHandle）は null を返す可能性があります。

CiscoTermEvFilter

boolean　`getSnapshotEnabled ()`

端末の CiscoTermSnapshotEv および CiscoTermSnapshotCompletedEv の有効または無効のステータスを返します。

void　`setSnapshotEnabled (boolean enabled)`

CiscoTermSnapshotEv の有効または無効のステータスを設定します。無効な場合、CiscoTermSnapshotEv および CiscoTermSnapshotCompletedEv はアプリケーションに送信されません。

boolean　`getRTPKeyEvEnabled ()`

CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv の有効または無効のステータスを返します。

void　`setRTPKeyEvEnabled (boolean enabled)`

CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv の有効または無効のステータスを設定します。

CiscoTerminal

void　**`createSnapshot ()`** throws InvalidStateException

このは、端末における現在のアクティブ コールのセキュリティ状態を格納した CiscoTermSnapshotEv を生成します。このメソッドにアクセスするには、端末が CiscoTerminal.IN_SERVICE 状態になっており、CiscoTermEvFilter.setSnapshotEnabled() が True に設定されている必要があります。

CiscoMediaTerminal

void	**register**(CiscoMediaCapability[] capabilities, int[] supportedAlgorithms) CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このインターフェイスではセキュアなメディアでの動的登録が可能です。アプリケーションがこのメソッドを呼び出さない場合、メディアはセキュアでないモードで終端されます。
void	**register**(java.net.InetAddress address, int port, CiscoMediaCapability[] capabilities, int[] algorithmIDs) CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このインターフェイスではセキュアなメディアでの静的登録が可能です。アプリケーションがこのインターフェイスを登録しない場合、メディアのセキュリティは確保されません。AlgorithmIDs はこの CTIPort がサポートする SRTP アルゴリズムを示します。AlgorithmID は、CiscoSupportedAlgorithm のいずれか 1 つだけになります。

CiscoRouteTerminal

void	register (CiscoMediaCapability)[] capabilities, int registrationType, int[] algorithmIDs CiscoRouteTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。デフォルトでは、メディアはセキュアでないモードで終端されます。AlgorithmIDs はこの CTIPort がサポートする SRTP アルゴリズムを示します。AlgorithmID は、CiscoSupportedAlgorithm のいずれか 1 つだけになります。

CiscoSupportedAlgorithm の定数

AES_128_COUNTER

SIP REFER または REPLACE

REFER は RFC 3515 で定義されている SIP メソッドです。REFER メソッドは、受信者（Request-URI によって識別される Referee）に対して、要求で提供されたコンタクト情報を使用して第三者（ターゲットと呼ばれる）にコンタクトするよう指示します。この REFER メソッドを使用すると、REFER を送信した端末（Referrer）はリクエストの結果を受け取ることができます。

Cisco Unified Communications Manager は Back-To-Back User Agent（B2BUA）として、内部ダイアログと外部ダイアログの両方で、Referee の代わりにインバウンド REFER を処理します。REFER の結果として、Cisco Unified Communications Manager は Referee と Refer To Target の間にコールを作成します。Referrer と Referee の間に既存のコールが存在する場合は、REFER が完了した後に Referrer 側でコールがドロップされます。

REPLACES 機能は既存の SIP ダイアログを新しいダイアログに置き換えます。SIP ダイアログが 2 つの SIP ユーザ エージェント間のコールであるのに対し、Cisco Unified Communications Manager ダイアログは半コール（コールレッグ）です。REPLACES 機能は REFER または INVITE によって開始されます。Cisco Unified Communications Manager は REPLACES ヘッダーの受信者の代わりに REPLACES 要求を処理します。この要求は新しいダイアログに関連付けられ、REPLACES ヘッダーで指定されている既存のダイアログ（コール）の相手側に代わりにコールに参加したい通話者が要求者になります。Cisco Unified Communications Manager は REPLACES ヘッダーで指定されているダイアログ（コール）の接続を解除して、要求者を接続します。

JTAPI は、Cisco Unified Communications Manager の REFER 機能と REPLACE 機能によって発生するコール イベントを JTAPI コール モデルでモデル化するように拡張されており、これらのコール イベントをアプリケーションで処理することを可能にします。JTAPI には REPLACES 要求を使用して REFER または REFER/INVITE を開始するためのインターフェイスはありませんが、これらのコール イベントを適切に処理することは可能です。

これら 2 つの機能には下位互換性があります。JTAPI は REFER/REPLACE によって発生したイベントを CAUSE_NORMAL で提供します。機能固有の原因は、新しいインターフェイスである CiscoCallEv.getCiscoFeatureReason() から取得できます。

（注） このインターフェイスは現行機能と新機能の両方に関して機能固有の原因を提供しますが、将来のリリースでは下位互換性はなくなる予定です。アプリケーションでこのインターフェイスを使用する場合は、将来的な下位互換性の問題を回避するために、デフォルトの処理を実装する必要があります。

次のセクションでは、SIP の REFER/REPLACE に関するインターフェイスの変更点について説明します。

REFER/REPLACE に対して提供される CAUSE

JTAPI は、REFER/REPLACES に起因するイベントに対して、CAUSE_NORMAL を提供します。アプリケーションでは CiscoCallEv.getCiscoFeatureReason() を使用して機能固有の原因を取得してください。

CiscoCallEv で提供されるインターフェイス

このインターフェイスは、JTAPI コール イベントに CiscoFeatureReason を提供します。転送などの既存の機能は、現状どおり元の CiscoCallEv.getCiscoCause() インターフェイスから CiscoCause を受け取ります。このインターフェイスは転送の REASON_TRANSFER を提供します。

```
com.cisco.jtapi.extensions
Interface CiscoCallEv
```

int　　**getCiscoFeatureReason()**

このインターフェイスは Cisco Unified Communications Manager 機能原因を返します。

Interface CiscoFeatureReason

JTAPI は、機能に起因する Call イベントで CiscoFeatureReason を提供します。CiscoFeatureReason は、Cisco Unified Communications Manager の既存の機能に対してだけでなく新規の機能に対しても提供されます。REFER および REPLACES の機能では、原因は REASON_REFER および REASON_REPLACES になります。このインターフェイスは、今後導入される新機能に対して新しい原因を提供するため、下位互換性はありません。

CiscoFeatureReason を使用するアプリケーションでは、今後のリリースで新しい原因コードが返される可能性があることを考慮してください。また、アプリケーションの下位互換性を保つためにはデフォルトの動作を実装する必要があります。

CiscoFeatureReason を使用するアプリケーションでは、今後のリリースで新しい原因コードが返される可能性があることを考慮してください。また、下位互換性を保つためにはデフォルトの動作を実装する必要があります。

Public interface CiscoFeatureReason

static int　`REASON_REFER`

REFER に対して Cisco Unified Communications Manager から送信されたイベントに対して返される原因。

static int　`REASON_REPLACE`

REPLACE に対して Cisco Unified Communications Manager から送信されたイベントに対して返される原因。

SIP 3XX リダイレクション

SIP リダイレクト サーバは SIP 要求を受信すると 3xx（リダイレクション）応答を返して、別の SIP アドレスにコンタクトするようクライアントに指示します。この機能拡張では、RFC 3261 に準拠した Cisco Unified Communications Manager リダイレクション（3xx）コール制御プリミティブがサポートされます。Cisco Unified Communications Manager リダイレクション プリミティブは SIP 3xx 応答を処理して、3xx 応答に含まれる各コンタクト アドレスに対して順次ハントを実行します。Cisco Unified Communications Managerまた、この操作の結果として発生する機能対話も処理します。Cisco Unified JTAPI は、このプリミティブの結果として Connection と TerminalConnection がいつ作成され、破棄されたかを示す新しい原因コードをすべての CallEv で公開します。

リダイレクション プリミティブがターゲットをハントしているときに JTAPI Redirect や CallForwardNoAnswer などの機能対話が発生した場合は、lastRedirectAddress が変更されることがあります。ターゲットが応答せず、Cisco Unified Communications Manager リダイレクトがコールの制御を引き継いで次のターゲットにコールを送信する場合、lastRedirectAddress は最初に SIP 3xx 応答を返した通話者に設定されます。

SIP 3xx 応答に Diversion ヘッダーが含まれている場合、3xx プリミティブは lastRedirectParty に対して Diversion ヘッダーの最初の値を使用し、JTAPI アプリケーションは Diversion ヘッダー要素を lastRedirectAddress として認識します。

JTAPI では、下位互換性を維持するために、CM_REDIRECTION という原因を含む新しい API、CiscoCallEv.getCiscoFeatureReason() を CiscoCallEv インターフェイスで公開しています。

（注） アプリケーションを開発する際には、この API から新しい機能固有の原因コードが返される可能性があることを考慮し、認識不能な原因コードに対してはデフォルトの動作を実行するようにしてください。

次のセクションでは、SIP 3XX Redirection に関するインターフェイスの変更点について説明します。

Public interface CiscoFeatureReason

static int　**REASON_CM_REDIRECTION**

この原因は、イベントが Cisco Unified Communications Manager の CM_REDIRECTION プリミティブからの 3xx 応答の結果であることを示します。

CiscoCallEv

int　`getCiscoFeatureReason()`

このイベントに対する機能固有の原因。アプリケーションは、未知の原因を処理してデフォルト動作を提示できる必要があります。これは将来新しい原因が追加され、このインターフェイスの下位互換性がなくなる可能性があるためです。

端末とアドレスの制限

この機能拡張では、管理者が Cisco Unified Communications Manager Administration の制限リストに特定の端末とアドレスのセットを追加した場合、それらの端末とアドレスをアプリケーションで制御または監視することが制限されます。

管理者はデバイス上の特定の回線（特定の端末上のアドレス）を制限リストに追加できます。Cisco Unified Communications Manager Administration の制限リストに端末を追加した場合は、JTAPI でもその端末のすべてのアドレスが制限付きとしてマークされます。この設定が完了した後にアプリケーションを起動した場合は、CiscoTerminal.isRestricted() インターフェイスと CiscoAddress.isRestricted(Terminal) インターフェイスを確認することで、特定の端末またはアドレスが制限されているかどうかを知ることができます。共用回線の場合は、CiscoAddress.getRestrictedAddrTerminals() インターフェイスを照会することで、特定のアドレスがいずれかの端末で制限されているかどうかを確認できます。

アプリケーションの起動後に回線（端末上のアドレス）が制限リストに追加された場合は、CiscoAddrRestrictedEv がアプリケーションに送信されます。そのアドレスにオブザーバがある場合は CiscoAddrOutOfService がアプリケーションに送信されます。制限リストから回線が削除されると、CiscoAddrActivatedEv がアプリケーションに送信されます。そのアドレスにオブザーバがある場合は CiscoAddrInServiceEv がアプリケーションに送信されます。アプリケーションが制限リストに含まれるアドレスにオブザーバを追加しようとすると、PlatformException がスローされます。アドレスが制限される前に追加されたオブザーバはそのまま残りますが、そのアドレスが制限リストから削除されない限り、これらのオブザーバでイベントを取得できません。アプリケーションからアドレスのオブザーバを削除することもできます。

アプリケーションの起動後にデバイス（端末）が制限リストに追加された場合は、CiscoTermRestrictedEv がアプリケーションに送信されます。その端末にオブザーバがある場合は CiscoTermOutOfService がアプリケーションに送信されます。端末が制限リストに追加されると、JTAPI でもその端末に属するすべてのアドレスが制限され、CiscoAddrRestrictedEv がアプリケーションに送信されます。制限リストから端末が削除されと、CiscoTermActivatedEv と、対応するアドレスの CiscoAddrActivatedEv がアプリケーションに送信されます。アプリケーションが制限リストに含まれる端末にオブザーバを追加しようとすると、PlatformException がスローされます。端末が制限される前に追加されたオブザーバはそのまま残りますが、その端末が制限リストから削除されない限り、これらのオブザーバでイベントを取得することはできません。

アプリケーションの起動後に共用回線が制限リストに追加された場合は、CiscoAddrRestrictedOnTerminalEv がアプリケーションに送信されます。そのアドレスにアドレス オブザーバがある場合は、その端末の CiscoAddrOutOfServiceEv がアプリケーションに送信されます。すべての共用回線が制限リストに追加された場合は、最後の 1 つが追加された時点で、CiscoAddrRestrictedEv がアプリケーションに送信されます。アプリケーションの起動後に制限リストから共用回線が削除された場合は、CiscoAddrActivatedOnTerminalEv がアプリケーションに送信されます。そのアドレスにオブザーバがある場合は、その端末の CiscoAddrInServiceEv がアプリケーションに送信されます。制御リストに含まれるすべての共用回線が制限リストから削除された場合は、最後の 1 つが削除された時点でアプリケーションに CiscoAddrActivatedEv が送信され、端末上のすべてのアドレスが InService イベントを受信します。

制御リストに含まれるすべての共用回線が制限リストに追加されている場合、アプリケーションがオブザーバを追加しようとすると、PlatformException がスローされます。一部の共用回線だけが制限リストに追加されている場合、アプリケーションがそのアドレスにオブザーバを追加すると、制限されていない回線だけがイン サービスになります。

アドレスまたは端末が制限リストに追加されてリセットされたときにアクティブ コールが存在していた場合は、接続と TerminalConnections が接続解除されます。

アドレスと端末が制限リストに 1 つも追加されていない場合、この機能は JTAPI の以前のバージョンと下位互換性を持ち、新しいイベントはアプリケーションに配信されません。

次のセクションでは、アドレスおよび端末の制限に関するインターフェイスの変更点について説明します。

CiscoTerminal

boolean	**isRestricted()** 端末が制限されているかどうかを示します。端末が制限されている場合、その端末に関連付けられたすべてのアドレスも制限されます。端末が制限されている場合は true を返します。端末が制限されていない場合は false を返します。

CiscoAddress

javax.telephony.Terminal[]	**getRestrictedAddrTerminals()** このアドレスが制限されている端末の配列を返します。制限されている端末がない場合、このメソッドは null を返します。 共用回線の場合、端末上の少数の回線が制限されている可能性があります。このメソッドは、このアドレスが制限されているすべての端末を返します。アプリケーションは、制限された回線のコール イベントは受信できません。制限された回線が他の制御デバイスとのコールに関わっている場合、制限された回線用の外部 Connection が作成されます。
boolean	**isRestricted**(javax.telephony.Terminal terminal) この端末上のいずれかのアドレスが制限されている場合、true を返します。この端末上のアドレスが制限されていない場合、false を返します。

```
public interface CiscoRestrictedEv extends CiscoProvEv {
    public static final int ID = com.cisco.jtapi.CiscoEventID.CiscoRestrictedEv;

    /**
     * The following define the cause codes for restricted events
     */

    public final static int CAUSE_USER_RESTRICTED = 1;

    public final static int CAUSE_UNSUPPORTED_PROTOCOL = 2;

    }
```

これは制限されたイベントの基底クラスであり、すべての制限されたイベントの原因コードを定義します。CAUSE_USER_RESTRICTED は、端末またはアドレスが制限とマークされていることを示します。CAUSE_UNSUPPORTED_PROTOCOL は、制御リスト内のデバイスが Cisco Unified JTAPI でサポートされていないプロトコルを使用していることを示します。SIP を実行している既存の Cisco Unified IP 7960 フォンおよび 7940 フォンはこれに該当します。

CiscoAddrRestrictedEv

public interface **CiscoAddrRestrictedEv** extends CiscoRestrictedEv。アプリケーションは、回線または関連するデバイスが Cisco Unified Communications Manager Administration から制限されたときに、このイベントを受信します。制限された回線では、アドレスがアウト オブ サービスになり、再び有効になるまでイン サービスに戻りません。アドレスが制限されている場合は、addCallObserver および addObserver によって例外がスローされます。共用回線では、いくつかの回線だけが制限されて残りは制限されなかった場合、例外はスローされませんが、制限された共用回線はイベントを受け取りません。すべての共用回線が制限された場合は、オブザーバを追加すると例外がスローされます。オブザーバを追加した後にアドレスが制限された場合、アプリケーションは CiscoAddrOutOfServiceEv を受信し、アドレスが有効にされるとイン サービスになります。

CiscoAddrActivatedEv

public interface **CiscoAddrActivatedEv** extends CiscoProvEv。アプリケーションは、回線または関連するデバイスが制御リストに含まれており、Cisco Unified Communications Manager Administration の制限リストから削除されたときに、このイベントを受信します。該当アドレスにオブザーバが存在する場合、アプリケーションは CiscoAddrInServiceEv を受信します。オブザーバが存在しない場合、アプリケーションはオブザーバの追加を試みることが可能で、アドレスはイン サービス状態になります。

CiscoAddrRestrictedOnTerminalEv

public interface **CiscoAddrRestrictedOnTerminalEv** extends CiscoRestrictedEv。ユーザが制御リストに共有アドレスを持っており、いずれかの回線を制限リストに追加した場合、このイベントが送信されます。getTerminal() インターフェイスは、アドレスが制限される端末を返します。getAddress() インターフェイスは、制限されるアドレスを返します。

javax.telephony.Address	**`getAddress()`**
javax.telephony.Terminal	**`getTerminal()`**

CiscoAddrActivatedOnTerminal

public interface **CiscoAddrActivatedOnTerminalEv** extends CiscoProvEv。共用回線または共用回線を持つデバイスを制限リストから削除すると、このイベントが送信されます。getTerminal() インターフェイスは、アドレスに追加される端末を返します。getAddress() インターフェイスは、新しい端末が追加されるアドレスを返します。

javax.telephony.Address	**`getAddress()`**
javax.telephony.Terminal	**`getTerminal()`**

CiscoTermRestrictedEv

public interface **CiscoTermRestrictedEv** extends CiscoRestrictedEv。アプリケーションは、アプリケーションの起動後にデバイスが Cisco Unified Communications Manager Administration から制限リストに追加されたときに、このイベントを受信します。アプリケーションは、制限された端末やその端末のアドレスに関するイベントは受信できません。InService 状態にある端末が制限された場合、アプリケーションはこのイベントを受信し、端末およびそれに対応するアドレスはアウト オブ サービス状態になります。

CiscoTermActivatedEv

public interface **CiscoTermActivatedEv** extends CiscoRestrictedEv。

javax.telephony.Terminal　**getTerminal()**

有効化されて制限リストから削除された端末を返します。

CiscoOutOfServiceEv

static int　**CAUSE_DEVICE_RESTRICTED**

デバイスが制限されたためにイベントが送られたかどうかを示します。

static int　**CAUSE_LINE_RESTRICTED**

回線が制限されたためにイベントが送られたかどうかを示します。

CiscoCallEv

static int　**CAUSE_DEVICE_RESTRICTED**

デバイスが制限されたためにイベントが送られたかどうかを示します。

static int　**CAUSE_LINE_RESTRICTED**

回線が制限されたためにイベントが送られたかどうかを示します。

Unicode のサポート

Cisco Unified Communications Manager リリース 5.0 は、Unicode 対応の IP フォンで Unicode の表示名をサポートします。表示名には ASCII 名と Unicode 名を設定できます。JTAPI は、すべての名前を Unicode 形式と ASCII 形式で受け取り、アプリケーションが表示名を Unicode で取得できるように、2 つの新しいインターフェイス getCurrentCalledPartyUnicodeDisplayName および getCurrentCallingPartyUnicodeDisplayName を提供します。また、コール プログレス中に Unicode 表示名を取得する機能もあります。

JTAPI は、デバイス登録イベントとデバイス イン サービス イベントの中で、アプリケーションによって制御される IP フォンのエンコード機能を CTI から受け取ります。また、ロケールおよび言語グループ情報をデバイス情報の応答で受け取ります。そして、IP フォンのロケール、代替スクリプト、および Unicode 機能を取得するためのインターフェイスをアプリケーションに提供します。CiscoTerminal および CiscoTermInServiceEv インターフェイスが拡張され、CiscoTerminal がイン サービス状態のときに、アプリケーションの制御リストにある電話にこれらの情報を提供します。

JTAPI は、コール内のすべての通話者の代替スクリプト情報を受け取り、現在の発信者と現在の着信者の言語グループを取得するためのインターフェイスを、アプリケーションに提供します。これらの情報を取得するための 2 つのインターフェイス getCurrentCallingPartyLanguageGroup および getCurrentCalledPartyLanguageGroup が、CiscoCall で使用できます。また、アプリケーションは、現在の発側アドレスと着側アドレスの表示名を ASCII 形式と Unicode（UCS-2）形式の両方で受信します。

JTAPI の Unicode サポートには次の変更も含まれます。

- CiscoCall インターフェイスの変更
- CiscoLocales インターフェイスの変更
- CiscoTerminal / CiscoTerminalInServiceEv インターフェイスの変更

リリース 5.0 にアップグレードした後は、アプリケーションでユーザ名 / パスワードの再設定が必要になる可能性があります。

次のセクションでは、Unicode のサポートに関するインターフェイスの変更点について説明します。

CiscoCall インターフェイスの変更

次に示す CiscoCall の新しいメソッドにより、アプリケーションは Unicode 表示名と対応するロケールを取得できます。

```
/**
     * This interface returns the unicode display name of the current called party
     * in the call.
     */
    public String getCurrentCalledPartyUnicodeDisplayName();

    /**
     * This interface returns the locale of the current called party unicode
     * display name. CiscoLocales interface lists the supported locales.
     */
    public int getCurrentCalledPartyUnicodeDisplayNamelocale();

    /**
     * This interface returns the unicode display name of the current calling party
     * in the call.
     */
    public String getCurrentCallingPartyUnicodeDisplayName ();

    /**
     * This interface returns the locale of the current called party
     * unicode display name
     */
    public int getCurrentCallingPartyUnicodeDisplayNamelocale();
```

CiscoLocales

CiscoLocales インターフェイスは、Cisco Unified JTAPI でサポートされているすべてのロケールを列挙します。

（注） 最新のリリースでサポートされているすべてのロケールのリストは、CiscoLocales の man ページを参照してください。

```
public interface CiscoLocales
{
    public static final int LOCALE_ENGLISH_UNITED_STATES;
    public static final int LOCALE_FRENCH_FRANCE;
    public static final int LOCALE_GERMAN_GERMANY;
    public static final int LOCALE_RUSSIAN_RUSSIA ;
    public static final int LOCALE_SPANISH_SPAIN ;
    public static final int LOCALE_ITALIAN_ITALY ;
    public static final int LOCALE_DUTCH_NETHERLAND ;
    public static final int LOCALE_NORWEGIAN_NORWAY ;
    public static final int LOCALE_PORTUGUESE_PORTUGAL;
    public static final int LOCALE_SWEDISH_SWEDEN ;
```

```
    public static final int LOCALE_DANISH_DENMARK
    public static final int LOCALE_JAPANESE_JAPAN;
    public static final int LOCALE_HUNGARIAN_HUNGARY ;
    public static final int LOCALE_POLISH_POLAND ;
    public static final int LOCALE_GREEK_GREECE ;
    public static final int LOCALE_TRADITIONAL_CHINESE_CHINA;
    public static final int LOCALE_SIMPLIFIED_CHINESE_CHINA;
    public static final int LOCALE_KOREAN_KOREA;
}
```

CiscoTerminalInServiceEv インターフェイス

int　**getLocale()**

このメソッドはこの端末に関連付けられた現在のロケール情報を返します。

int　**getSupportedEncoding ()**

このメソッドは、この端末が Unicode をサポートしている場合に true を返します。

CiscoTerminal インターフェイス

int　**getLocale()**

このメソッドはこの端末に関連付けられた現在のロケール情報を返します。このメソッドにアクセスするには、CiscoTerminal が CiscoTerminal.IN_SERVICE 状態になっている必要があります。

int　**getSupportedEncoding ()**

このメソッドは、この Terminal の Unicode 機能を返します。このメソッドにアクセスするには、CiscoTerminal が CiscoTerminal.IN_SERVICE 状態になっている必要があります。

getSupportedEncoding () は、CiscoTerminal で定義されている次のいずれかの結果を返します。

```
/**
  * Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
  * for this Terminal is UNKNOWN
*/
 public final static int UNKNOWN_ENCODING = 0;

/**
  * Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
  * for this is NOT_APPLICABLE.
  * This is valid for only CiscoMediaTerminals and RoutePoints
*/
 public final static int NOT_APPLICABLE = 1;

/**
  * Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE> for this
  * Terminal is ASCII and this terminal supports only ASCII_ENCODING
*/
 public final static int ASCII_ENCODING = 2;
```

```
/**
  * Indicates the <Code>CiscoTerminal.getSupportedEncoding ()</CODE>
  * for this Terminal is UCS2UNICODE_ENCODING
 */
 public final static int UCS2UNICODE_ENCODING = 3;
```

Linux、Windows および Solaris へのインストール

この機能は、Windows、Linux、および Solaris プラットフォームにおける Cisco Unified JTAPI クライアントのインストール/アンインストール手順を統一できます。以前のリリースでは、Windows プラットフォーム用の Cisco Unified JTAPI クライアント インストーラしか提供されていなかったため、Linux および Solaris マシンへのインストールは手動で行う必要がありました。

Cisco Unified JTAPI Install は InstalledShieldMultiPlatform（ISMP）を使用して開発されています。Linux 版と Solaris 版のインストーラはバイナリ ファイル（.bin）として提供され、Windows 版のインストーラは実行可能（.exe）ファイルとして提供されます。これら 3 種類のインストーラはすべて、Cisco Unified Communications Manager のプラグイン ページから入手できます。

Windows プラットフォームの JTAPI クライアントについては、このリリースへのアップグレードおよびこのリリースからのダウングレードに関して次のような変更点があります。

- 新しいインストーラは、以前のリリースへのダウングレードをサポートしていません。以前のバージョンの JTAPI クライアントを再インストールする場合は、事前に JTAPI を手動でアンインストールする必要があります。
- 新しいインストーラを使用して以前のリリースからアップグレードする場合は、現在のバージョンの JTAPI クライアントをアンインストールするよう求めるメッセージが表示され、以前のバージョンをアンインストールした後にインストールが開始されます。

サイレント インストール呼び出しを使用すると、アプリケーションのインストーラに JTAPI Installer を組み込むことができます。インストーラをサイレント モードで起動するアプリケーションでは、次のいずれかのコマンドを使用できます。

- Windows：　CiscoJTAPIClient.exe –silent
- Linux：　CiscoJTAPIClient-linux.bin –silent
- Solaris（Sparc）：CiscoJTAPIClient-solarisSparc.bin –silent
- Solaris（X86）：CiscoJTAPIClient-solarisX86.bin –silent

コマンドラインから JTAPIInstaller を実行する必要があるアプリケーションの場合は、コマンド プロンプトから次のいずれかのコマンドを使用できます。

- Windows：　CiscoJTAPIClient.exe -console
- Linux：　CiscoJTAPIClient-linux.bin –console
- Solaris（Sparc）：CiscoJTAPIClient-solarisSparc.bin –console
- Solaris（X86）：CiscoJTAPIClient-solarisX86.bin –console

コマンドを使用する場合は、文字入力ベースのメニューによってインストール手順が示されます。GUI を使用してインストールする場合と同じオプションを使用できます。このタイプのインストールは、Linux のような非 GUI プラットフォームに最も適しています。

Linux プラットフォームで新規インストールまたはアップグレード/ダウングレードを実行すると、JTAPIInstaller がインストール先フォルダを自動的に検出してサイレント インストールを実行します。ただし、システムに以前のバージョンが存在する場合は、インストーラがアプリケーション パスを判断できないため、デフォルトの場所にフォルダが作成され、それらのフォルダにアプリケーションがインストールされます。

JTAPIInstaller の詳細については、第 4 章「Cisco Unified JTAPI のインストール」を参照してください。

Linux へのインストール

インストーラで新規インストールまたはアップグレードを実行すると、インストール先フォルダが自動的に検出されて、サイレント インストールが実行されます。ただし、システムに以前のバージョンが存在する場合は、インストーラがアプリケーション パスを判断できないため、デフォルトの場所にフォルダが作成され、そのフォルダにアプリケーションがインストールされます。

下位互換性

この機能には下位互換性はありません。CiscoJtapiClient インストールのコマンド ラインまたはサイレント インストール機能では、インストールのコマンド呼び出しを変更する必要があります。

サイレント インストール

JTAPIInstaller でサイレント インストール呼び出しを実行すると、アプリケーションのインストールに JTAPIInstaller を組み込むことができます。サイレント インストールを呼び出す方法については、セクション（5.1.2.14.1）で説明しています。システムに SD よりも前のバージョンが存在し、SD バージョンへのアップグレード操作が発生した場合、新しいインストーラによって、SD よりも前のバージョンのアンインストーラが自動的に呼び出されます。アプリケーションで JTAPI の SD よりも前のバージョンをサイレント モードでアンインストールする必要がある場合、次の方法でコマンド ラインからインストーラを起動する必要があります。CiscoJtapiClient.exe -W newversion.silent=1。このメソッドは、Windows プラットフォームにだけ適用されます。新しい SD インストーラのインストール前、つまりインストーラを実行する前に、アプリケーションで JTAPI のバージョンを検出する必要がある場合は、次のコマンドを使用して検出を行ってください。

CiscoJtapiClient.exe silent -W newversion.check=1 goto showversion このコマンドをコマンド ラインから実行すると、インストーラにより、現在のディレクトリに jtapiversion.txt というファイルが作成されます。これにより JTAPI バージョンのインストーラが a.b(c.d) 形式になります。このコマンドは、すべてのクライアント プラットフォーム（Linux、Solaris、Windows）に適用されます。

コマンドライン呼び出し

このモードは、Linux アカウントなど、GUI をサポートしていないシステムに JTAPI をインストールする場合に便利です。すべてのインストール手順が文字入力ベースのメニューによって示され、ユーザはインストール時の条件に基づいて一連の入力を行うように求められます。このモードでは GUI ベースのインストーラで提供されているその他のオプションもすべて使用できます。

JTAPI クライアント インストーラ

ISMP インストーラを使用したインストール時に、システムに JTAPI クライアント インストーラの前のリリースが存在する場合、ISMP インストーラにより、レジストリから自動的にこのリリースが検出され、前のリリースのアンインストーラが起動されます。ISMP インストーラが次の操作に進んだ場合、ユーザは前のリリースのアンインストール操作が正常に完了したことを確認する K 要があります。JTAPI クライアント インストーラの前のリリースのアンインストール中に何らかのエラーが発生しても、ISMP インストーラはそのエラーを検出できないため、インストールを続行します。

ISMP インストーラには、実際のインストールを続行する前に、古い JTAPI クライアント インストーラをアンインストールする上記のメソッドが導入されていますが、ユーザは次の手順に従うことをお勧めします。

手順

ステップ 1　[プログラムの追加と削除] に進みます。

ステップ 2　前のリリースのアンインストールを実行します。

この手順を推奨するのは、ユーザが直接現在のリリースのインストーラを起動しないようにするためです。インストーラは前のリリースのアンインストーラを呼び出します。アンインストールが完了すると、ユーザはシステムの再起動を今すぐ行うか、後で行うかを指定するように求められます。ユーザが今すぐに再起動する方を選択すると、サーバはリブートされ、同時にインストーラは新しいバージョンをインストールしようとします。前のリリースのアンインストーラと現在のリリースのインストーラは、異なる種類の IS/ISMP で作成されており、この 2 つのインストーラはリンクしていないため、この問題が発生します。

また、ISMP インストーラは、前のリリースのダウングレード操作もサポートしていません。

下位互換性

この機能には下位互換性はありません。

JRE 1.2 および JRE 1.3 のサポートの削除

このリリースの Cisco JTAPI クライアントは JRE 1.4 のみをサポートします。インターフェイスの変更はありませんが、JRE 1.2 と 1.3 はサポートされません。この変更の目的は、JDK 1.4 以降のみで利用可能な QoS をサポートすることです。また、jtapi.jar に含まれる Cisco の暗号化ファイルを利用するには JRE 1.3 以降が必要になります。このファイルは TCP を介して CTIManager にパスワードを送信するときに強力なパスワード暗号化アルゴリズムを提供します。JTAPI は、この機能の一部として、パスワードを送信する前に IMS（Identity Management System：Cisco Unified Communications Manager のコンポーネント）が提供する API を呼び出してパスワードを暗号化します。

また、JRE 1.4 を使用することで、JDK 1.4 に追加された新しい API を Cisco Unified JTAPI で利用できるようになります。以前のバージョンの JRE を使用するアプリケーションで Cisco Unified JTAPI を使用するには、JDK 1.4 をインストールする必要があります。

（注）　JTAPI アプリケーションへのインターフェイスに変更はありませんが、JTAPI.jar には RSA jsafe.jar（3.3）ファイルと Apache log4j-1.2.8.jar ファイルが含まれています。これらのバージョンの jsafe.jar（バージョン 3.3）および log4j-1.2.8.jar と互換性がない jar ファイルをアプリケーションで使用している場合、どちらのファイルがクラスパスで先に指定されているかによって、JTAPI またはアプリケーションが正常に動作しないことがあります。

また、この移行により、JTAPIPreferences とサンプル アプリケーションの MS-JVM に対する依存関係も解消されました。[JTAPI Preferences] ダイアログボックスの [詳細設定] タブに次の 2 つの設定パラメータが新しく追加されました。

- JTAPI Post Condition Timeout
- Use Progress As Disconnected

下位互換性

この機能には下位互換性はありません。

スーパープロバイダーと変更通知

このリリースの JTAPI に含まれるスーパープロバイダー関連の機能拡張では、主に次の点が変更されています。

プロバイダーがオープンされた後に Cisco Unified Communications Manager Administration で「Superprovider privilege」が無効に設定されると、そのことが CTI 変更通知イベントを通じて JTAPI に通知され、オープンされているデバイスのうち制御リストにないものがすべてクリーン アップされます。

JTAPI は、この変更を「CiscoProviderCapabilityChangedEvent」を使用してアプリケーションに通知します。この新しいイベントはフラグが変更されたときに発行され、フラグが有効になったか無効になったかを示します。制御リストにないデバイスがスーパープロバイダー モードでオープンされて制御リストに追加された場合、JTAPI はそのデバイスを自身の制御リストに追加します。

- このフラグが有効な「CiscoProviderCapabilityChangedEvent」を通常のアプリケーションが受信した場合は、スーパープロバイダー特権が付与されていることを意味しているので、このアプリケーションは自身の制御リストにないデバイスの取得を開始できます。
- スーパープロバイダー アプリケーションが、Superprovider フラグが無効な「CiscoProviderCapabilityChangedEvent」を受信した場合は、このアプリケーションのスーパープロバイダー特権が解除されたことを意味します。その後、次の一連のイベントが発生します。
  - Provider Out of Serivce（OOS）イベントがアプリケーションに配信され、そのアプリケーションが取得 / オープンしたデバイスがすべてクローズされます。
  - 取得 / オープンされたデバイスのうち制御リストにないすべてのデバイスについて CiscoTermRemovedEv がアプリケーションに配信されます。
  - JTAPI が通常のユーザとして CTI への再接続に成功すると、Provider inService イベントがアプリケーションに配信されます。
  - デバイスと回線の情報がアプリケーションに配信されます。
  - プロバイダーが OOS に移行する前にオープンしていたすべての制御対象デバイスについて CiscoTermCreatedEv がアプリケーションに配信されます。
- Cisco Unified Communications Manager Administration で「パーク DN モニタリング」フラグが変更された場合、JTAPI は「CiscoProviderCapabilityChangedEvent」を使用してアプリケーションに通知します。
  - このフラグが有効なイベントを受信したアプリケーションは、パーク DN を制御するための登録機能を実行します。
  - このフラグが無効に設定されているイベントをアプリケーションが受信した場合、JTAPI は再び「CiscoProviderCapabilityChangedEvent」を使用してアプリケーションに通知し、すべてのパーク DN アドレスをクローズします。
- Cisco Unified Communications Manager Administration で「change calling party number」フラグが変更されると、JTAPI はそのことを「CiscoProviderCapabilityChangedEvent」を使用してアプリケーションに通知します。
  - このフラグが有効なイベントを受信したアプリケーションは、発信者番号を変更できるようになります。
  - このフラグが無効なイベントを受信したアプリケーションは、発信者番号を変更できません。

  アプリケーション側でも、このフラグが無効な場合は発信者番号を変更しないようにする必要があります。

- スーパープロバイダーが制御リストにないデバイスをオープン / 取得した後に Cisco Unified Communications Manager Administration でそのデバイスが削除された場合、JTAPI は端末オブジェクトをクローズして、そのデバイスの CiscoTermRemovedEvent をアプリケーションに送信します。

インターフェイスの変更

スーパープロバイダーと変更通知の拡張の一部として、JTAPI では次の API をアプリケーションに公開しています。その結果、スーパープロバイダーのための JTAPI 実装と特定のプロバイダー機能の処理が変更されています。このリリースの JTAPI に含まれるスーパープロバイダー関連の機能拡張としては、JTAPI の QBE インターフェイス、JTAPI の動作に関する変更、アプリケーションに公開される新しい API があります。

JTAPI は、CiscoProviderCapabilityChangedEv を次の形式でアプリケーションに配信します。アプリケーションでは、JTAPI から送信されるこの新しいイベントを受信し、処理できるようにしてください。

```
public interface CiscoProviderCapabilityChangedEv {
    public CiscoProviderCapabilities getCapability ();
}
```

CiscoProviderCapabilities には、プロバイダーの発信者変更特権を設定するための次の新しいメソッドがあります。

```
public boolean canModifyCallingParty();
public void setCanModifyCallingParty(boolean value);
```

CiscoProviderCapabilityChangedEv は、適切なフラグ値とともにアプリケーションに配信されます。

その後、次の一連のイベントが発生します。

- JTAPI は、プロバイダー OOS イベントをアプリケーションに送信し、デバイス / 回線 OOS を制御リスト内のオープンされているデバイスと回線に送信します。
- 次に、JTAPI は CTI への再接続を試みます。
  - 再接続に成功した場合、JTAPI はプロバイダー inService イベントを送信し、以前にオープンされていた制御リスト内のすべてのデバイスを再びオープンします。
  - 再接続に失敗した場合、JTAPI はプロバイダーをシャットダウンし、ProviderClosedEvent を送信します。
- スーパープロバイダー特権が追加された場合、JTAPI は CiscoProviderCapabilityChangedEv を適切なフラグ値とともにアプリケーションに送信します。
  - MonitorParkDN フラグが有効になっている場合、JTAPI はパーク DN 監視フラグを true に設定して CiscoProviderCapabilityChangedEv を送信します。
  - MonitorParkDN フラグが無効になっている場合、JTAPI はパーク DN 監視フラグを false に設定して CiscoProviderCapabilityChangedEv を送信します。

    また、JTAPI はすべてのパーク DN アドレスをクローズし、CiscoAddrRemovedEv をアプリケーションに配信します。
- ModifyCgPn フラグが変更された場合、JTAPI はプロバイダー オブジェクト内でフラグを設定します。このフラグがリダイレクトのシナリオの中で調べられ、それに応じてアプリケーションによる発信者の変更権限が許可または禁止されます。

  また、JTAPI は CgPn を変更するためのフラグを設定して CiscoProviderCapabilityChangedEv を配信します。

CiscoProvider インターフェイス

boolean	`hasSuperproviderChanged()` スーパープロバイダー特権が変更されたかどうかをアプリケーションに知らせます。
boolean	`hasModifyCallingPartyChanged()` ModifyCgPn 特権が変更されたかどうかをアプリケーションに知らせます。
boolean	`hasMonitorParkDNChanged()` パーク DN 監視特権が変更されたかどうかをアプリケーションに知らせます。

下位互換性

この機能には下位互換性はありません。

代替スクリプトのサポート

一部の機種の IP フォンでは、電話機上で設定可能なロケールに対応するデフォルト スクリプト以外の代替言語スクリプトがサポートされています。たとえば、日本語ロケールには 2 つの表記スクリプトがあります。一部の機種では Katakana スクリプト（デフォルト）しかサポートされていませんが、その他の機種ではデフォルト スクリプトと Kanji スクリプト（代替スクリプト）の両方がサポートされています。アプリケーションは電話機に表示用のテキスト情報を送信できるため、電話機がどの代替スクリプトをサポートしているかを知る必要があります。

新しい getAltScript() メソッドは、監視対象のデバイスの代替スクリプト情報を提供します。現在存在する代替スクリプトは日本語ロケールの Kanji だけです。

JTAPI では、代替スクリプト情報を提供する CiscoTerminal 用の新しいメソッドが提供されています。

java.lang.String	`getAltScript()` 現在サポートされている代替スクリプトは日本語ロケールの Kanji だけです。空の文字列が返された場合、代替スクリプトが設定されていないか、端末が代替スクリプトをサポートしていないことを表します。

下位互換性

代替スクリプトの機能は JTAPI の下位互換性に影響を与えることはありません。

半二重メディアのサポート

現在、JTAPI メディア イベントの CiscoRTPInputStarted、CiscoRTPOutputStarted、CiscoRTPInputStopped、および CiscoRTPOutputStopped では、メディアが半二重（受信のみ / 送信のみ）か全二重（受信と送信の両方）かは示されません。

この機能拡張により、JTAPI メディア イベントでこの情報を提供する機能が追加されます。JTAPI では、上記のメディア イベントでメディアが半二重か全二重かを問い合せるインターフェイスが提供されています。

半二重メディアのサポート機能は JTAPI の下位互換性に影響を与えることはありません。

新しいインターフェイス getMediaConnectionMode() が Cisco Unified JTAPI の RTP イベントに追加されています。このインターフェイスはメディアに応じて次の値を返します。

- CiscoMediaConnectionMode.NONE
- CiscoMediaConnectionMode.RECEIVE_ONLY
- CiscoMediaConnectionMode.TRANSMIT_ONLY
- CiscoMediConnectionMode.TRANSMIT_AND_RECEIVE

CiscoRTPInputStarted/StoppedEv は、RECEIVE_ONLY および TRANSMIT_AND_RECEIVE だけを返す必要があります。通常、NONE または TRANSMIT_ONLY が返されることはありません。返された場合は、イベントを無視するか、エラーを記録してください。

CiscoRTPOutputStarted/StoppedEv は、TRANSMIT_ONLY および TRANSMIT_AND_RECEIVE だけを返す必要があります。通常、値 NONE または RECEIVE_ONLY が返されることはありません。返された場合は、イベントを無視するか、エラーを記録してください。

CiscoMediaOpenLogicalChannedEv は、RECEIVE_ONLY および TRANSMIT_AND_RECEIVE だけを返す必要があります。通常、値 NONE または TRANSMIT_ONLY が返されることはありません。返された場合は、イベントを無視するか、エラーを記録してください。

public interface **CiscoRTPInputStartedEv**

int　`getMediaConnectionMode()`

CiscoMediaConnectionMode を返します。

public interface **CiscoRTPOutputStartedEv**

int　`getMediaConnectionMode()`

CiscoMediaConnectionMode を返します。

public interface **CiscoRTPInputStoppedEv**

int　`getMediaConnectionMode()`

CiscoMediaConnectionMode を返します。

public interface **CiscoRTPOutputStoppedEv**

int　　getMediaConnectionMode()

CiscoMediaConnectionMode を返します。

ネットワーク アラート

以前のリリースの Cisco JTAPI（Cisco JTAPI バージョン 1.4(x.y)）では、クラスタ外のアドレスへのコールを開始すると、遠端のアドレスに CallCtlConnNetworkReachedEv イベントと CallCtlConnNetworkAlertingEv イベントが配信されていました。

最近のバージョンの Cisco Unified Communications Manager（4.0 以降）および Cisco Unified JTAPI（2.0）では、これらのイベントは配信されていません。これらのバージョンでは、遠端アドレスの CallCtlConnection が OFFERED 状態から ESTABLISHED 状態に移行しました。以前のバージョンの Cisco Unified JTAPI では、「オーバーラップ送信」がオフの状態でゲートウェイ経由のコールが発信されたときに、遠端アドレスに対して CallCtlConnOfferedEv および CallCtlConnEstablishedEv が配信されていました。CallCtlConnNetworkReachedEv および CallCtlConnNetworkAlertingEv イベントはアプリケーションに配信されませんでした。

Cisco Unified Communications Manager 4.0 および 4.1 では、ネットワーク イベントを受信するために、ゲートウェイに対して設定されているルート パターンの「オーバーラップ送信を許可」フラグ、または jtapi.ini の「AllowNetworkEventsAfterOffered」パラメータをオンにする必要がありました。

Cisco Unified Communications Manager リリース 5.0 では、「オーバーラップ送信を許可」フラグが有効な場合、ゲートウェイの向こう側にある遠端アドレスの ConnCreatedEv、CallCtlConnNetworkReachedEv、CallCtlConnNetworkAlertingEv、および CallCtlConnEstablishedEv がアプリケーションに配信されます。

「オーバーラップ送信を許可」フラグが有効でない場合は、ゲートウェイの向こう側にある遠端アドレスの ConnCreatedEv、CallCtlConnOfferedEv、CallCtlConnNetworkReachedEv、CallCtlConnNetworkAlertingEv、および CallCtlConnEstablishedEv がアプリケーションに配信されます。

（注）　Cisco Unified Communications Manager リリース 5.0 では、「AllowNetworkEventsAfterOffered」は使用できません。上記のイベントは jtapi.ini のパラメータ設定に関係なく配信されます。

下位互換性

この機能には下位互換性はありません。

Linux の自動アップデータ

Linux ベースの JTAPI クライアント マシンでこの機能をサポートするために、アップデータのインターフェイスに次の変更が加えられました。このインターフェイスでは、アプリケーションは、コンポーネント名、プロバイダーの IP アドレス、ユーザ名、パスワードを提供する必要があります。アプリケーションは、コンポーネントをダウンロードする URL を指定する必要はありません。これは、Cisco Unified Communications Manager Administration のリリースごとに URL が異なる場合に、アップデータ アプリケーションに生じる問題を回避するために実現されました。

「Replace()」という名前の新しい API が、コンポーネント インターフェイスの一部として組み込まれています。この API を使用すると、古いコンポーネントを新しくダウンロードしたコンポーネントに簡単に置換できます。次のセクションでは、新しいインターフェイスへの変更後のアップデータの動作を定義しています。新しいアップデータは次のように動作します。

- API 署名は、古いものと同じものを使用する。
- 新しいバージョンの jar ファイルとして、アプリケーションの現在のフォルダに newjtapi.jar を作成する。
- 指定されたクラスパスの component.temp というファイルに現在の jtapi.jar をコピーする。
- 現在の jar ファイルを新しい jar ファイルに置換する。この処理が終わると、現在の jar ファイルは component.temp になり、新しい jar ファイルは jtapi.jar になります。アプリケーションでは、URL の取得を、アプリケーションで URL を指定する方法、または CiscoProvider で提供されている新しいインターフェイスを使用して URL を問い合わる方法のいずれかで行う、古いコンポーネント インターフェイスを使用することもできます。URL 情報を取得する必要がある API については、この機能のインターフェイスの概要に記載しています。この処理は、Unix と Windows の両方でサポートされています。

下位互換性

この機能には下位互換性はありません。

コールの選択状態

機能によって、または手動でコールを選択した場合、Cisco Unified JTAPI により CiscoTermConnSelectChangedEv イベントが送信されます。アプリケーションはこのイベントを受信すると、TerminalConnection.getSelectStatus() を使用して正確なコールの選択状態を取得できます。TerminalConnection.getSelectStatus() を呼び出すと、次の 3 つの状態のいずれかが返されます。

- CiscoTerminalConnection.CISCO_SELECTEDNONE：この選択状態は、コールが選択されていないことを示します。
- CiscoTerminalConnection.CISCO_ SELECTEDLOCAL：この選択状態は、コールが TerminalConnection で選択されていることを示します。
- CiscoTerminalConnection.CISCO_ SELECTEDREMOTE：コールがその共用回線で選択されている場合、Passive TerminalConnection がこの選択状態を受け取ります。

下位互換性

この機能には下位互換性はありません。

JTAPI バージョン情報

Cisco Unified Communications Manager Administration のリリース 5.0 に接続するには、JTAPI クライアントを、Cisco Unified Communications Manager Administration リリース 5.0 に付属した JTAPI の新しいバージョンにアップグレードする必要があります。JTAPI のバージョンは 3.0(X.Y) の形式で提供されます。この、X と Y は、サブリリースごとに異なります。アプリケーションは、JTAPI の古いリリースと接続できません。

コール転送

Cisco Unified JTAPI では、JTAPI 仕様に適合したコール転送機能の設定がサポートされています。Cisco Unified JTAPI 実装では、すべての転送特性はサポートされず、アドレスの FORWARD_ALL 属性だけがサポートされます。CallControlAddress オブジェクトの setForwarding、getForwarding および cancelForwarding のメソッドをアプリケーションによって呼び出すことが可能ですが、CallControlForwarding 命令は FORWARD_ALL のものだけが可能です。

コール パーク

Cisco Unified JTAPI は、ユーザのコール パークとの対話をサポートし、アプリケーションに適切なイベントを報告します。コールが IP フォンからパークされると、パーク アドレスに属する Connection は、Disconnected 状態に移行し、関連する TerminalConnection は、Dropped 状態に移行します。パーク番号に対してキュー状態の新規 Connection が作成されます。

コールのパークされたアドレスだけをアプリケーションが監視している場合、すべての既存の Connection は解除され、TerminalConnections が破棄されてコールは Invalid 状態に移行します。

パークの取得

コールが IP フォンからパークされると、電話機にパーク番号が表示されます。どの端末からでも、パーク番号をダイヤルすればコールのパークを解除できます。コールのパークが解除されると、パークを解除したアドレスに接続する新しいコールが作成されます。Queued 状態にある元のコールのパーク番号に対する CallControlConnection は、Disconnected 状態に移行します。

パーク リマインダ

パークされたコールが指定時間内に取得されないと、コールのパークされたアドレスにリマインダ コールが返され、パーク番号 Connection は Disconnected 状態に移行します。コールは再接続され、確立された状態に移行します。コールのパークされたアドレスに Talking 状態の TerminalConnection が作成されます。

パーク DN モニタ

Cisco Unified JTAPI アプリケーションは、コールがパークまたはパーク解除されたときにイベントを受信するように登録できます。この機能に登録すると、プロバイダー オブザーバに CiscoProvCallParkEv イベントが配信されます。この機能に登録するには、ユーザの Call Park Retrieval Allowed フラグがオンになっている必要があります。このフラグには Cisco Unified Communications Manager Administration のユーザ設定でアクセスできます。この機能に登録すると、クラスタ内のデバイスでコールがパークまたはパーク解除されるたびに、アプリケーションが CiscoProvCallParkEv イベントを受信します。

この機能の登録および登録解除は、次の新しいインターフェイスで行えます。

```
public interface CiscoProvider {
public void registerFeature ( int featureID ) throws
    InvalidStateException, PrivilegeViolationException;
public void unregisterFeature ( int featureID ) throws
    InvalidStateException;
}
```

featureID は CiscoProvFeatureID.MONITOR_CALLPARK_DN です。

CiscoJtapiExceptions

Cisco Unified JTAPI では、アプリケーションに CTI 生成のエラー コードが通知されます。これらのコードは、CTIManager に例外またはエラーがある場合に返されます。CTI で返されたエラー コードは、それぞれアプリケーションに伝搬されます。アプリケーションでは、例外オブジェクトの getErrorCode() メソッドの呼び出しによるエラー コードの抽出、getErrorName() メソッドの呼び出しによる CTI エラー コードの取得、およびメソッド getErrorDescription() の呼び出しによるエラー記述の取得が可能です。

getErrorName(int errorCode) メソッドと getErrorDescription (int errorCode) メソッドは推奨しません。今後のリリースで削除される予定です。Cisco では、アプリケーション ユーザがこれらのメソッドを使用しないことを推奨します。

Cisco Unified JTAPI インストールの国際対応

Cisco Unified JTAPI では、JTAPI のインストールとユーザ プリファレンスの UI に複数の言語がサポートされています。JTAPI が起動すると、インストール用の言語を選択するオプションが表示されます。言語を選択すると、その後のインストールの指示は選択した言語で表示されます。最初のオプションは常に英語です。語句が英語以外の言語に欠落している場合、指示はデフォルトの英語になります。詳細については、第 4 章「Cisco Unified JTAPI のインストール」を参照してください。

コールのクリア

Cisco Unified JTAPI アプリケーションでは、アクティブ コールを破棄せずにファントム コールをクリアできます。CiscoAddress には clearCallConnections メッセージがあり、これにより、Cisco Unified Communications Manager（以前の Cisco Unified Call Manager）上にアクティブ コールがないときにアプリケーションによってコールをクリアできます。

デバイス復旧

Cisco Unified JTAPI では、デバイスの自動復旧がサポートされています。

電話機のデバイス復旧

Cisco Unified IP Phone 7960 などのデバイスについては、リホーム機能がデバイス ファームウェアの一部に組み込まれています。プライマリ Cisco Unified Communications Manager の障害時に、コール上に Communications Manager がなくなると、電話機からバックアップ Cisco Unified Communications Manager への接続が試みられます。この移行は、「CTIManager の障害」（P.3-103）で説明するアウト オブ サービスおよびイン サービスのイベントの形式でアプリケーションに送信されます。

CTI Port や CTI RoutePoint などのファームウェアのない仮想デバイスでは、CTIManager または Cisco Unified JTAPI によってフェールオーバーが実行されます。

CTI RoutePoint

Cisco Unified Communications Manager サーバの障害時には、CTI RoutePoint のデバイス プール管理で定義された Cisco Unified Communications Manager サーバ グループから CTIManager がデバイスを復旧させます。プライマリ Cisco Unified Communications Manager サーバが復旧すると、CTIManager はそのプライマリ Cisco Unified Communications Manager 上のデバイスの復旧を試みます。このリホームは、デバイス上にコールがない場合に行われます（物理デバイスと同様）。

CTIManager の障害時には、Cisco Unified JTAPI がバックアップ CTIManager 上のデバイスを復旧させます。CiscoAddrOutOfServiceEv イベントと CiscoAddrInServiceEv イベントによって、デバイスのアベイラビリティがアプリケーションに通知されます。

CTI Port

アプリケーションによって登録される CTI Port には、電話機に似たメカニズムがあります。CTIPort へのシグナリングを処理している Cisco Unified Communications Manager に障害が起きると、CTIManager はこのデバイス用のデバイス プール管理に従ってそのサービスを復旧させます。CTIManager の障害時には、CTI Port がバックアップ CTIManager に接続されると、Cisco Unified JTAPI によって CTI Port が再登録されます。CTI Port の復旧後、CiscoAddrOutOfServiceEv イベントおよび CiscoTermOutOfServiceEv イベントと対応するイン サービスのイベントが送信されます。

これらのデバイスに対するメディア ストリーミングは、アプリケーションによって制御され、ポートがアウト オブ サービスの場合でもストリーミングは継続します。新しいコールを受け入れる準備が完了するまで、デバイスに対して新しいコールが発行されないように、アプリケーション側で処理します。

ディレクトリ変更通知

アプリケーションでは、デバイス追加、ユーザ制御リストからの削除、および Cisco Unified Communications Manager データベースからのデバイスの削除に関する通知を非同期に要求します。また、アプリケーションは回線変更およびデバイスに関する通知を受信します。この通知は Cisco Unified JTAPI に送信され、それぞれ AddressObserver と TerminalObserver 上の CiscoAddrCreatedEv、CiscoAddrRemovedEv、CiscoTermCreatedEv、および CiscoTermRemovedEv によってアプリケーションに伝搬されます。

（注）　回線変更通知を受信するために、CTIPort と CTIRoutePoint に対して必ずデバイスを登録してください。

呼び出し音の無効化または有効化

CiscoAddress の拡張により、デバイス上のすべての回線に対する呼び出し音のステータスをアプリケーションによって設定できます。管理用ウィンドウやその他のウィンドウから呼び出し音の設定が変更されても、イベントは生成されません。

転送と会議の拡張

JTAPI の転送イベントと会議イベントには、理解しにくい面があります。これは、参加者が一方のコールからもう一方のコールに移動する際に、JTAPI では、一方のコールから通話者を削除し、もう一方のコールに通話者を追加することで、このアクションを表現しているためです。これは、実際には通話者が移行中であるときに、コールから破棄されたという通知がアプリケーション上で受信されるため、混乱を招く場合があります。Cisco Unified JTAPI 実装では、アプリケーションによるこの機能の処理を容易にする、いくつかの追加イベントが定義されています。

転送

転送機能では、1 つのコール（転送コール）の参加者が別のコール（最後のコール）に転送されます。コールの参加者を移動すると、コール転送に関連付けられた Connection が DISCONNECTED 状態に移行し、これらの参加者の新しい Connection が最後のコールに作成されます。同様に、転送コールに関連するすべての TerminalConnection が Dropped 状態に移行し、最後のコール内に作成されます。Cisco の拡張機能では、定義で、転送に関連するイベントの開始と終了をマークします。

CiscoConnection.getConnectionID() メソッドを使用して、新旧の Connection に関する CiscoConnectionID を取得することで、新たに作成された Connection オブジェクトと古い Connection オブジェクトを関係付けることができます。Connection が一致する場合は、CiscoConnectionID.equals() メソッドを使用して比較したときに CiscoConnectionID オブジェクトが一致します。

CiscoTransferStartEv

このイベントでは、転送動作が開始されたことが示され、後に続くイベントはこの動作に関連しています。具体的には、Connection と TerminalConnection は両方とも削除され、転送結果として追加されます。

アプリケーションによって、転送に関与する 2 つのコール（転送コールと最後のコール）、およびこのイベントからの転送コントローラ情報を取得できます。JTAPI アプリケーションで転送コントローラを監視していない場合、このイベントでは転送コントローラ情報を使用できません。

CiscoTransferEndEv

このイベントでは、転送動作が終了したことが示されます。このイベントの受信後、アプリケーションでは、関与するすべての通話者が転送され、すべての Connection と TerminalConnection が最後のコールに移動されたと想定できます。

転送シナリオ

次のシナリオでは、転送に関与する 3 つの通話者をそれぞれ A、B、C で表します。

B が転送コントローラとなるコンサルト転送

コンサルト転送では、アプリケーションでコールを別のアドレスにリダイレクトし、転送者はリダイレクトの前に転送先と「コンサルト（情報交換）」できます。

- コール Call1 で A が B をコールする。
- B が応答し、コール Call2 で C とコンサルトする。
- B はコール Call2 をコール Call1 に転送する。

このタイプの転送を行うには、次の JTAPI メソッドを使用します。

- Call2.setTransferEnable(true)（このオプションのメソッドは、コール オブジェクト内で転送がデフォルトで有効であることを示します。）
- Call2.consult(TermConnB, C)
- Call1.transfer(Call2)

（注）　コンサルト転送時には、Call2.transfer(Call1) ではなく Call1.transfer(Call2) でコールが転送されます。

表 3-5 に、A と B のオブザーバが CiscoTransferStartEv と CiscoTransferEndEv の間に受信するコア イベントを示します。

表 3-5　A と B のオブザーバに対するコア イベント

原因メタ イベント	コール	イベント	フィールド
META_UNKNOWN	Call1	CiscoTransferStartEv	transferredCall=Call2 finalCall=Call transferController= TermConnB
META_CALL_TRANSFERRING	Call1	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv C CallInvalidEv C	
META_UNKNOWN	Call2	CallObservationEndedEv	
META_UNKNOWN	Call1	CiscoTransferEndEv	transferredCall=Call2 FinalCall=Call1 transferController= TermConnB

A が転送コントローラとなる任意転送

任意転送では、コールを作成した方法に関係なく、あるコールを別のコールに転送できます。コンサルト転送とは異なり、最初にコンサルト メソッドを使用してコールの 1 つを作成する必要はありません。

- コール Call1 で A が B をコールする。

- A は Call1 を保留にする。
- コール Call2 で A が C をコールする。
- A は Call1 を Call2 に転送する。

このタイプの転送を行うには、次の JTAPI メソッドを使用します。

- コール Call1 を最後のコール Call2 に転送する Call2.transfer(Call1)、または
- コール Call2 を最後のコール Call1 に転送する Call1.transfer(Call2)

Call1.transfer(Call2) が呼び出されたと想定し、表 3-6 に、A と C のオブザーバが CiscoTransferStartEv と CiscoTransferEndEv の間に受信するコア イベントを示します。

表 3-6　　A と C のオブザーバに対するコア イベント

原因メタ イベント	コール	イベント	フィールド
META_UNKNOWN	Call1	CiscoTransferStartEv	transferredCall=Call2 finalCall=Call1 transferController= TermConnB
META_CALL_TRANSFERRING	Call1	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	
META_CALL_TRANSFERRING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv C CallInvalidEv C	

会議

JTAPI では、2 つのコールで会議を行うときには、一方のコールのすべての通話者をもう一方のコールに移動することが規定されています。所属していた通話者が移動され、それ以降、無効になるコールは、「マージ」コールまたは「コンサルト」コールと呼びます。マージされた通話者の移動先のコールを、これ以降「最終」コールと呼びます。通話者がマージ コールから最後のコールに移動するとき、アプリケーションでは、すべての通話者がマージ コールから破棄されたことを示すイベント、およびこれらの通話者が最終コールに再現されたことを示すイベントが受信されます。

新たに作成された Connection オブジェクトと古い Connection オブジェクトの相関を取るには、CiscoConection.getConnectionID() メソッドを使用して、すべての新旧の Connection に対する CiscoConnectionID オブジェクトを取得します。Connection が一致する場合は、CiscoConnectionID.equals() メソッドを使用して比較したときに CiscoConnectionID オブジェクトが一致します。

会議サポートは、次のメソッドに存在します。

- javax.telephony.callcontrol.CallControlCall.conference(Call)
- javax.telephony.callcontrol.CallControlCall.getConferenceController()
- javax.telephony.callcontrol.CallControlCall.getConferenceEnable()
- javax.telephony.callcontrol.CallControlCall.setConferenceController(TerminalConnection)
- javax.telephony.callcontrol.CallControlCall.setConferenceEnable(boolean)

Cisco の拡張

Cisco Unified JTAPI 実装では、会議の開始と終了を通知する、CiscoConferenceStartEv と CiscoConferenceEndEv の 2 つのイベントが追加されます。これらのイベントは、会議の起動時と終了時に送信されます。これらのイベントから、最後のコール、マージされた会議（コンサルト）コールおよび 2 つの制御対象の TerminalConnection（HELD 状態と TALKING 状態）へのハンドルが取得されます。

CiscoConferenceStartEv

このイベントは、call1.conference(call2) が呼び出されるか、または IP フォン上で 2 度目に [Conference（会議）] ボタンが押された場合に送信されます。ConferenceStartEv では、マージ プロセスの開始が通知されます。このイベントの後に、Cisco Unified Communications Manager の会議プロセスによって反映される一連のマージ イベントが続きます。

CiscoConferenceEndEv

このイベントは、ConferenceStartEv の送信後、マージ プロセスの終了時に送信されます。これにより、コンサルト（またはマージ）コールの最終電話会議へのマージ完了が通知されます。マージ コールは INVALID 状態になり、ObservationEndedEv がコール オブザーバに送信されます。

CiscoCall.setConferenceEnable()

Cisco Unified JTAPI 実装では、CiscoCall.setConferenceEnable() メソッドと CiscoCall.setTransferEnable() メソッドを使用することにより、会議または転送のどちらの機能を使用してコンサルト コールを開始するかを制御します。どちらの機能も明示的に有効にしていない場合は、デフォルトで転送が使用されます。

会議シナリオ

次のシナリオでは、呼び出し可能な 2 種類の典型的な会議について説明します。

会議制御に B を使用したコンサルト会議

次の一連の手順では、このシナリオを一般的に説明します。

- A が B にコール（Call 1）する。
- B が応答する。

- B は C にコンサルト（Call 2）する。

  setConferenceEnable()

  ```
  call2.consult(tc, C)
  ```

- C が応答する。
- B が会議を開催する。

  ```
  Call1.conference(Call2)
  ```

（注）　コンサルテーション後、元のコールの conference() メソッドを呼び出し、会議を開催する必要があります。コンサルト コール オブジェクト内で会議を呼び出すと、例外がスローされます。

会議制御に B を使用した任意会議

次の一連の手順では、このシナリオを一般的に説明します。

- A が B にコール（Call 1）する。
- B が応答する。
- B はコールを保留する。
- B は C にコール（Call 2）する。
- C が応答する。
- B が会議を開催する。

  ```
  Call1.conference(Call2)
  ```
  または
  ```
  Call2.conference(Call1)
  ```

会議イベント

表 3-7 に、Call1.Conference(Call2) 呼び出し時の、一連のコア イベント（コール制御および Cisco の拡張）を示します。

表 3-7　一連のイベント

原因メタ イベント	コール	イベント	フィールド
META_UNKNOWN	Call1	CiscoConferenceStartEv	consultCall = Call2 finalCall = Call1 conferenceController= TermConnB
META_CALL_MERGING	Call1	CallCtlTermConnTalkingEv B	
META_CALL_MERGING	Call1	ConnCreatedEv C ConnConnectedEv C CallCtlConnEstablishedEv C TermConnCreatedEv C TermConnActiveEv C CallCtlTermConnTalkingEv C	
META_CALL_MERGING	Call2	TermConnDroppedEv B CallCtlTermConnDroppedEv B ConnDisconnectedEv B CallCtlConnDisconnectedEv B	

表 3-7　一連のイベント （続き）

原因メタ イベント	コール	イベント	フィールド
META_CALL_MERGING	Call2	TermConnDroppedEv C CallCtlTermConnDroppedEv C ConnDisconnectedEv C CallCtlConnDisconnectedEv C CallInvalidEv C	consultCall=Call2 finalCall=Call1 conferenceController= TermConnB
META_UNKNOWN	Call2	CallObservationEndedEv	
META_UNKNOWN	Call1	CiscoConferenceEndEv	

転送と会議の拡張

コール転送に関与するすべての通話者には、CiscoTransferStartEv と CiscoTransferEndEv が送信されます。会議通話に関与するすべての通話者には、CiscoConferenceStartEv と CiscoConferenceEndEv が送信されます。コール転送では、最初のコールへの Connection の破棄と 2 番目のコールへの Connection の作成の、2 つのイベントも生成されます。Cisco Unified Communications Manager リリース 3.1 では、これらのイベントの順序が変更されています。最初に最後のコールの Connection が作成され、次にコンサルト コールの Connection が破棄されます。

（注）　Cisco Unified Communications Manager リリース 3.0 では、コールの転送に関与するすべての通話者に、これらのイベントが送信されるとは限りませんでした。

（注）　Cisco Unified Communications Manager リリース 3.0 から 3.1 へアプリケーションを移植する際、転送用の CiscoTransferStartEv と CiscoTransferEndEv の間、または会議用の CiscoConferenceStartEv と CiscoConferenceEndEv の間に発生するイベントの順序にアプリケーションが依存しないようにしてください。

メディアのないコンサルト

メディア パスを確立せずに転送用のコンサルト コールが実行されていることを、アプリケーションから Cisco Unified Communications Manager に通知できます。実際の転送の前にエージェントが使用可能かどうかを判別するために、コンサルタント コールが実行されている場合は、中継コールのためにメディア パスを確立する必要はありません。CiscoConsultCall インターフェイスで定義されている consultWithoutMedia メソッドでは、メディア パスを確立せずにコンサルタント コールが作成されます。

（注）　コンサルト コールは、転送だけが可能です。これは会議には使用できません。

メディア終端の拡張

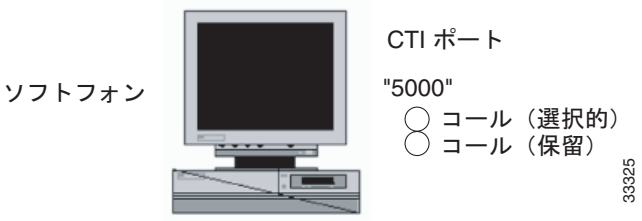
メディア終端機能を利用すると、オーディオやビデオなどのコールのベアラの、アプリケーションによる送信およびキャプチャが可能です。このアクションは、メディアの「再生と録音」または「ソースとシンク」と表現されることもあります。メディア終端は、コールのセットアップやティアダウンの詳細ではなく、コールのエンドポイント間を流れるデータに関与するため、コール制御とは区別されます。

たとえば、Automatic Call Distributor（ACD; 自動呼配送）では、コール制御を使用して使用可能なエージェント間でコールがルーティングされますが、メディアは終端されません。その一方で、Interactive Voice Response（IVR; 対話式音声自動応答）アプリケーションでは、コール制御を使用してコールの応答または接続解除を行い、メディア終端を使用して発信者側に対してサウンド ファイルを再生します。

メディア終端に単独で関与するテレフォニー アプリケーションはありませんが、この機能は常にコール制御と組み合せて使用されます。JTAPI 1.2 では、主にコール制御仕様が記述されており、メディア終端を必要とするアプリケーションには非常に限定的なサポートしか提供されません。Cisco Unified Communications ソリューションのプラットフォームでは、JTAPI 標準よりも高度なメディア終端がサポートされるため、Cisco Unified JTAPI 実装では、JTAPI を拡張してこの機能のフル サポートを追加しています。

Cisco Unified JTAPI では、Computer Telephony Integration（CTI）ポートを使用して、ソフトウェアベースのメディア終端が行われます。これらのポートには、コールの発信または受信に使用可能な 1 本以上の回線（ダイヤル可能な番号）があります。ただし、これらのポートにはメディアのソースとシンクを提供する制御アプリケーションが必要です。アプリケーションは、Cisco Unified Communications Manager を使用して、メディア終端ポートへの関与を登録します。これで、Cisco Unified Communications Manager からアプリケーションにこの仮想デバイスに関連するすべてのイベントが配送されます。Cisco Unified JTAPI では、CTI ポートは CiscoMediaTerminal と表現されます。図 3-6 に CTI ポートの構成を示します。CTI ポートの管理と構成についての詳細は、Cisco Unified Communications Manager Administration の情報を参照してください。

図 3-6　　CTI ポートの図



ソフトフォン アプリケーション（たとえば、PC が電話機として動作）を実装する場合は、Cisco Unified JTAPI アプリケーションを使用して CTI ポートを管理します。

Cisco Unified Communications Manager メディア エンドポイント モデル

エンドポイントは、Cisco Unified Communications ソリューションのプラットフォーム上で、IP フォンやゲートウェイなどのメディアを終端するエンティティを表します。1 つのエンドポイントから別のエンドポイントへのコールによって、2 つのエンドポイント間にメディアのフローが生じます。Cisco Unified Communications ソリューションのプラットフォーム内にあるすべてのエンドポイントにおいて、Real-Time Protocol（RTP）を使用して音声データが送信されます。たとえば、Cisco Unified Communications ソリューションの電話機とゲートウェイには、RTP スタックが組み込まれています。また、アプリケーションが Cisco Unified Communications ソリューションのシステム内でエンドポイントとして機能し、メディアを終端することも可能です。すべての Cisco Unified Communications ソリューションのエンドポイントにおいて RTP が使用されるため、アプリケーションにも RTP パケットを送信および受信する機能が必要です。

ペイロードとパラメータのネゴシエーション

ベアラ データおよびペイロードに加えて、各 RTP パケットにはヘッダーが格納されます。このヘッダーは、一連のパケットをメディア ストリームに再構成およびデコードする方法をエンドポイント側で決定する際に役立ちます。ただし、RTP では、あるストリームに対してどのペイロード タイプを使用するかをエンドポイントでネゴシエートする方法は用意されていません。たとえば、オーディオ データは、G.711 標準を使用してエンコードされます。また、RTP では、エンコード済みデータのサンプリング レートや各 RTP パケットで転送されるサンプル数などの固有のペイロード タイプ パラメータとネゴシエートする方法も提供されません。この代わりに、RTP は通常、エンドポイントでこれらのパラメータとネゴシエートする独自の方法を指定できる H.323 などの別のプロトコルと併用されます。このようなネゴシエーションはすべて、エンドポイント間で RTP パケットを送信する前に実行されます。

RTP ストリームに対するペイロードの選択とパラメータのエンコードは、エンドポイントではなく Cisco Unified Communications Manager が担当します。一般的な双方向のオーディオ通話には、次の 5 つの手順があります。

- 初期化
- ペイロード選択
- チャネル割り当ての受信
- 送受信の開始
- 送受信の停止

初期化

始動時には、各エンドポイントから Cisco Unified Communications Manager にそれぞれのメディア機能、つまり、G.711、G.723、G.729a などが通知されます。たとえば、最初に電話機がオンになったとき、または前の接続が切れてから電話機と Cisco Unified Communications Manager が再接続された後に、IP フォンが始動します。エンドポイントでは、1 つのペイロード タイプと別のペイロード タイプのプリファレンスを表現できませんが、パケット サイズなどの各ペイロード タイプの特定のパラメータを指定することは可能です。

エンドポイントで登録された機能リストは、排他的かつ不変です。G.711 と G.723 のサポートが可能とエンドポイントによって示されている場合、G.729a はサポートされないことを意味します。さらに、エンドポイントでサポートされる機能のリストを変更する場合は、エンドポイントを Cisco Unified Communications Manager との接続から解除し、再度初期化する必要があります。

JTAPI アプリケーションでは、Cisco Unified Communications Manager に CiscoMediaTerminal を登録して、この手順を実行します。CiscoMediaTerminal.register() メソッドでは、アプリケーションを使用して、Cisco Unified Communications Manager への登録のためのメディア機能オブジェクトの配列を提供できます。この手順では、Cisco Unified Communications Manager 設定内のデバイス設定に従って、アプリケーションが特定の電話番号に関するすべての発着コールのエンドポイントとして機能することが、Cisco Unified Communications Manager に通知されます。

ペイロード選択

双方向メディア ストリームが 2 つのエンドポイント間に作成される直前、たとえばコールがエンドポイントで応答されるとき、Cisco Unified Communications Manager によってメディア ストリームに対する適切なペイロード タイプ（コーデック）が選択されます。Cisco Unified Communications Manager により、コールに関与する双方のエンドポイントのメディア機能が比較され、使用に適した共通のペイロード タイプとペイロード パラメータが選択されます。

ペイロードを選択する際の基準には、エンドポイントの機能と場所が使用されますが、今後、この選択ロジックに他の基準が追加される可能性があります。エンドポイントは、コールごとのペイロード タイプの選択に動的に関与することはありません。

チャネル割り当ての受信

Cisco Unified Communications Manager では、2 つのエンドポイント間の RTP ストリームで共通のペイロード タイプが検出できる場合、各エンドポイントに対して、論理的な「受信チャネル」（エンドポイントがコールの RTP データを受信する固有の IP アドレスとポート）の作成が要求されます。この要求を受けて、各エンドポイントから Cisco Unified Communications Manager に IP アドレスとポートが返されます。

現在、IP フォンとゲートウェイに限ってこの手順が実行されます。Cisco Unified Communications Manager では、初期化中に固定 IP アドレスとポートを JTAPI アプリケーション側で指定する必要があります。したがって、JTAPI アプリケーションを使用して、同じエンドポイントに対する複数のメディア ストリームを同時に終端できません。アプリケーション側で複数のメディア ストリームを終端する場合、複数のエンドポイントを同時に登録する必要があります。

オープンしている受信チャネル要求に対してエンドポイントからの迅速な応答がない場合、Cisco Unified Communications Manager によってコールの接続が解除されます。JTAPI アプリケーションによる CiscoMediaTerminal の登録時には常に IP アドレスが提供されるため、アプリケーションで制御されるエンドポイントへのコールについては、接続が解除されません。ただし、Cisco Unified Communications Manager で、コールに関与する 2 つのエンドポイント間に共通のペイロード タイプが検出されない場合、Cisco Unified Communications Manager によってコールの接続が解除されます。

送受信の開始

Cisco Unified Communications Manager では、双方のチャネル情報を受信した後に、RTP ストリーム用に選択されたコーデック パラメータと、もう一方のエンドポイントの送信先アドレスが、各エンドポイントに通知されます。この情報は、各エンドポイントに対して、送信開始メッセージと受信開始メッセージの 2 つのメッセージで送信されます。

JTAPI アプリケーションでは、RTP データの送信と受信に必要なコーデック パラメータのすべてを含む CiscoRTPOutputStartedEv イベントと CiscoRTPInputStartedEv イベントを受信します。

JTAPI における QoS ベースライン化作業の一環として、CiscoRTPOutputStartedEv は getPrecedenceValue() API をアプリケーションに提供します。CTI により、値「音声コールの DSCP 値」が JTAPI に提供されます。アプリケーションではこの値を使用して、アプリケーションで開くメディア ストリームに対して DSCP 値を設定できます。

送受信の停止

保留や接続解除などの機能のために RTP ストリームを中断する必要がある場合、RTP データの送信と受信を停止する要求が Cisco Unified Communications Manager から各エンドポイントに対して送信されます。メディア フローの開始と同時に、送信停止と受信停止のメッセージが個別に送信されます。

JTAPI アプリケーションでは、CiscoRTPOutputStoppedEv と CiscoRTPInputStoppedEv を受信します。

Cisco MediaTerminal

JTAPI では、端末オブジェクトは、コールに対する論理エンドポイントを表し、データ（デジタルでエンコードされた音声サンプルなど）の送受信が可能であると想定されます。このため、Cisco Unified IP Phone は、JTAPI の端末として表現されます。ただし、ゲートウェイは、メディア終端を実行しますが、端末としては表現されません。CiscoMediaTerminal が、アプリケーションがメディア終端を担当する特殊なエンドポイントを表します。

CiscoMediaTerminal の使用に関連する手順は次の 4 つです。

- プロビジョニング
- 登録
- オブザーバの追加
- コールの受け入れ

プロビジョニング

CiscoMediaTerminal は実際のハードウェアの IP フォンやゲートウェイを表すことはありませんが、物理端末と同様に Cisco Unified Communications Manager でプロビジョニングされることに注意してください。IP フォンがデバイス ウィザードを使用して Cisco Unified Communications Manager データベースに追加されるのと同様に、同じ方法で CiscoMediaTerminal が追加されます。これにより Cisco Unified Communications Manager はアプリケーションのエンドポイントを電話番号およびコール転送などの他のコール制御プロパティに関連付けることができます。DeviceWizard では、CiscoMediaTerminal というデバイス タイプは存在しません。その代わりに、Cisco Unified Communications Manager には、アプリケーションの登録をサポートする複数のデバイス タイプがあります。JTAPI では、これらの各タイプが CiscoMediaTerminal として公開されます。現在、JTAPI の CiscoMediaTerminal を表すデバイス タイプは CTI ポートだけです。

次に、アプリケーションで制御されるエンドポイントとして使用するために CTI ポートをプロビジョニングする手順を示します。

手順

ステップ 1　Cisco Unified Communications Manager の設定ウィンドウ内で、デバイス ウィザードを使って [デバイス] > [電話] ウィンドウから、CTI ポートのデバイスを追加します。この CTI ポートのデバイス名には、対応する JTAPI の CiscoMediaTerminal の名前を指定します。

ステップ 2　[ユーザ] > [Global Directory] ウィンドウを使用して、新しい CTI ポート デバイスを、アプリケーションが [ユーザ] ウィンドウを使って制御するデバイスのリストに追加します。

詳細については、『*Cisco Unified Communications Manager Administration Guide*』を参照してください。

登録

メディア終端デバイスが Cisco Unified Communications Manager 内で正しくプロビジョニングされると、アプリケーションでは、Provider.getTerminal() メソッドまたは CiscoProvider.getMediaTerminal() メソッドを使用して、対応する CiscoMediaTerminal オブジェクトへの参照を取得できます。2 つのメソッドの違いは、CiscoProvider.getMediaTerminal() では CiscoMediaTerminal だけが返されるのに対し、Provider.getTerminal() では、物理的な IP フォンを表現するオブジェクトなど、プロバイダーに関連するすべての端末オブジェクトが返される点です。

Cisco Unified Communications Manager に対して特定のペイロード タイプの RTP ストリームを終端する意図を通知するには、CiscoMediaTerminal.register() を使用します。CiscoMediaTermina.register() メソッドには、IP アドレス、ポート番号、およびアプリケーションでサポートするコーデックのタイプとコーデック固有のパラメータを示す CiscoMediaCapability オブジェクトの配列を指定します。

IP アドレスとポートは、アプリケーションがメディア ストリームを受信できるアドレスを示します。次のコード例に、CiscoMediaTerminal を登録し、ポート番号 1234 のローカル アドレスにバインドする方法を示します。

```
CiscoMediaTerminal registerTerminal (Provider provider, String terminalName) {
  final int PORT_NUMBER = 1234;
  try {
   CiscoMediaTerminal terminal = provider.getTerminal (terminalName);
   CiscoMediaCapability [] caps = new CiscoMediaCapability [1];
   caps[0] = CiscoMediaCapability.G711_64K_30_MILLISECONDS;
   terminal.register (InetAddress.getLocalHost (), PORT_NUMBER, caps);
  }
  catch (Exception e) {
    return null;
  }
}
```

このコード例を機能させるには、指定したプロバイダーが IN_SERVICE である必要があります。さらに、このコードでは、定数 CiscoMediaCapability.G711_64K_30_MILLISECONDS が使用されることに注意してください。これは実際には、最大 30 ミリ秒の RTP パケット サイズを指定する、CiscoG711MediaCapability オブジェクトへの静的な参照を表します。これと他の共通のメディア形式は、CiscoMediaCapability クラスで定義されています。

CiscoMediaCapability クラスに示されていないメディア ペイロードを指定するには、2 つのオプションがあります。対象となるペイロード タイプが CiscoMediaCapability の既存サブクラスからの単純な派生の 1 つである場合、必要な処理はサブクラスの新規インスタンスの構築だけです。たとえば、最大 60 ミリ秒の RTP パケット サイズの G.711 ペイロードをアプリケーションでサポートできる場合、コンストラクタに 60 ミリ秒を指定するなどして CiscoG711MediaCapability オブジェクトを直接構築できます。

その一方で、対象となるペイロード タイプに一致する CiscoMediaCapability の既存のサブクラスがない場合には、CiscoMediaCapability クラスのインスタンスを直接構築します。CiscoMediaCapability の構築時に指定できる他のパラメータは、30 ミリ秒などの最大パケット サイズだけです。

次のコードに、カスタムのペイロード機能の登録方法を示します。

```
CiscoMediaTerminal registerTerminal (Provider provider, String terminalName) {
  final int PORT_NUMBER = 1234;
  try {
   CiscoMediaTerminal terminal = provider.getTerminal (terminalName);
   CiscoMediaCapability [] caps = new CiscoMediaCapability [1];
   caps[0] = new CiscoMediaCapability (
     RTPPayload.G728,
     30  // maximum packet size, in milliseconds
     );
   terminal.register (InetAddress.getLocalHost (), PORT_NUMBER, caps);
  }
  catch ( Exception e) {
    return null;
  }
}
```

CiscoMediaCapability オブジェクトの構築に使用するペイロード タイプのパラメータは、RTP ヘッダーのペイロード フィールドに対応します。このために、RTPPayload インターフェイスでは、既知のペイロード タイプが定義されています。

オブザーバの追加

RTP データを送受する場所と時間を示すイベントを受信するには、CiscoMediaTerminal に CiscoTerminalObserver を配置します。CiscoTerminalObserver では、新しいメソッドを定義することなく標準の JTAPI TerminalObserver インターフェイスが拡張されます。これは、RTP イベントの受信時にアプリケーションの関与を通知するマーカー インターフェイスを提供するものです。

（注） これは、CallObserver ではなく TerminalObserver であるため、Terminal.addCallObserver() メソッドではなく Terminal.addObserver() メソッドを使用して追加する必要があります。

さらに、CiscoMediaTerminal に関連付けられたアドレス オブジェクトに CallControlCallObserver を追加します。これにより、CiscoMediaTerminal への呼び出しコールがあるときに、アプリケーションに通知されることが保証されます。呼び出しコールを自動的に受け入れる通常の IP フォンとは異なり、CiscoMediaTerminal は、提供されるコールの受け入れ、接続解除（拒否）、またはリダイレクトを実行します。CallCtlConnOfferedEv は、Terminal オブジェクトではなく Address オブジェクト上に配置された CallControlCallObserver に対してだけ提示されるため、アプリケーションでは、その CallControlCallObserver を正しい場所に配置します。

（注） CallObserver インターフェイスだけでなく、CallControlCallObserver インターフェイスも確実に実装してください。CallCtlConnOfferedEv はコアの CallObserver インターフェイスだけを実装するオブザーバには配送されません。

コールの受け入れ

着信コールが CiscoMediaTerminal アドレスに到着した場合、TerminalConnection の作成前に CallControlConnection.accept() メソッドを使用してコールを受け入れる必要があります。このプロセスは、発信コールには適用されません。この Connection は、コールが電話番号確認の次へ進行するとすぐに、CallControlConnection.ESTABLISHED 状態になるからです。Connection が受け入れられると、発側の TerminalConnection に応答してメディア フローを開始します。発信側エンドポイントの機能に登録された機能に Cisco Unified Communications Manager を合わせることが可能である場合、アプリケーションによって RTP データの送受信を開始できるように、Cisco Unified Communications Manager からメディア フロー イベントが送信されます。

メディア フロー イベントの受信と応答

2 つのエンドポイント間にメディア ストリームが作成される場合は必ず、Cisco Unified Communications Manager から双方のエンドポイントに送信開始と受信開始のイベントが発行されます。JTAPI では、CiscoRTPOutputStartedEv イベントと CiscoRTPInputStartedEv イベントで送信開始と受信開始のイベントを表します。CiscoRTPOutputStartedEv.getRTPOutputProperties() メソッドでは CiscoRTPOutputProperties オブジェクトが返されます。アプリケーションでは、このオブジェクトから、コールのピア エンドポイントの送信先アドレス、およびペイロード タイプやパケット サイズなどのストリームに関する RTP プロパティを確認できます。同様に、CiscoRTPInputStartedEv.getRTPInputProperties() メソッドでは、アプリケーションに着信ストリームの RTP 特性を通知する CiscoRTPInputProperties オブジェクトが返されます。

メディアが流れている間は常に、CiscoMediaTerminal.getRTPOutputProperties() メソッドと CiscoMediaTerminal.getRTPInputProperties() メソッドから現在の CiscoRTPOutputProperties と CiscoRTPInputProperties も使用できます。CiscoMediaTerminal でメディアの送受信が想定されていない場合、これらのメソッドによって例外がスローされます。

Cisco Unified Communications Manager によって、たとえばコールの接続解除または保留の結果として、アプリケーションによるメディアの送受信が停止される場合は、CiscoRTPOutputStoppedEv イベントと CiscoRTPInputStoppedEv イベントが送信されます。これらのイベントは、2 つのエンドポイント間に存在する現在の RTP メディア ストリームをティアダウンする必要があることを意味します。

着信コール メディア フロー イベントの図

表 3-8 に、アプリケーションで制御されるエンドポイントにコールが提示される場合の Cisco Unified Communications Manager と JTAPI アプリケーション間のダイアログを示します。左の列のイベントはアプリケーションの CallObserver に送信される JTAPI イベント、右の列の要求はアプリケーションで呼び出されるメソッドを表します。

表 3-8　着信メディア フロー イベント

JTAPI イベント	方向	アプリケーションの要求
CallActiveEv ConnCreatedEv ConnProceedingEv CallCtlConnOfferingEv	→	
	←	CallControlConnection.accept ()
CallCtlConnAlertingEv TermConnCreatedEv TermConnRingingEv	→	
	←	TerminalConnection.answer ()
ConnConnectedEv CallCtlConnEstablishedEv TermConnTalkingEv CiscoRTPOutputStartedEv CiscoRTPInputStartedEv	→	
	←	CallControlConnection.disconnect ()
CiscoRTPOutputStoppedEv CiscoRTPInputStoppedEv TermConnDroppedEv CallCtlConnDisconnectedEv	→	

（注）　表 3-8 では、ローカル接続、つまり、アプリケーション エンドポイント向けの JTAPI イベントを示しています。実際の JTAPI メタ イベント ストリームには、発側の状態を記述したイベントが含まれます。

Cisco Unified Communications ソリューションの RTP 実装

Cisco Unified Communications ソリューションのアーキテクチャによって性能が改善されたため、Cisco Unified Communications ソリューションの電話とゲートウェイでは、RTP の一部の機能と、通常、RTP の関連付けられる Real-Time Control Protocol（RTCP）を実装していません。アプリケーションの互換性を確保するには、次の点を考慮する必要があります。

- RTCP はサポートされません。Cisco Unified Communications ソリューションのエンドポイントから RTCP メッセージが送信されず、エンドポイントに送信されるこの種のメッセージはすべて無視されます。
- Cisco Unified Communications ソリューションのエンドポイントでは現在、RTP ヘッダーの「synchronization source」（SSRC）フィールドは使用されていません。アプリケーションによって SSRC フィールドを使用した RTP ストリームを多重化しないでください。多重化すると、電話機とゲートウェイにおいて、メディアを正しくデコードおよび提示できません。

リダイレクト

JTAPI 1.2 では、CallControlConnection.redirect() メソッドの事前条件の 1 つとして、Connection が CallControlConnection.OFFERING または CallControlConnection.ALERTING のどちらかの状態にあることが規定されています。Cisco Unified JTAPI では、CallControlConnection.ESTABLISHED 状態の Connection もリダイレクトできます。

redirect() メソッドには、CiscoConnection インターフェイス内に次のオーバーロードされた形式があります。これにより、コールのリダイレクト中に障害が起きたときに、必要な動作をアプリケーションによって指定し、発信元検索スペースを指定したり、元の着信フィールドをリセットできます。

オーバーロードのリダイレクト メソッドにある次の INT パラメータの 1 つを CiscoConnection インターフェイスから渡すことで、アプリケーションに必要な動作を選択します。

- 障害時のリダイレクト ドロップ：コールが使用中または無効な送信先にリダイレクトされた場合、Cisco Unified Communications Manager では、リダイレクトが失敗した場合にコールを破棄するか、またはリダイレクト コントローラにコールを残す処理のどちらかを実行できます。この後、JTAPI アプリケーションを使用して、別の送信先にコールをリダイレクトするなどの修正処置ができます。リダイレクト モード パラメータには、次のオプションがあります。
  - CiscoConnection.REDIRECT_DROP_ON_FAILURE
  - CiscoConnection.REDIRECT_NORMAL
- コール サーチ スペース：リダイレクトでは、コール サーチ スペースのパラメータを使用して、使用する callingSearchSpace を指定します。アプリケーションでは、発側のコール サーチ スペースまたはリダイレクト コントローラのコール サーチ スペースのどちらかを使用できます。このシナリオには、次のパラメータ オプションがあります。
  - CiscoConnection.CALLINGADDRESS_SEARCH_SPACE
  - CiscoConnection.ADDRESS_SEARCH_SPACE
- 元の着信のリセット：着信アドレス オプションのパラメータを使用して、元の着信フィールドをリセットします。このシナリオには、次のオプションがあります。
  - CiscoConnection.CALLED_ADDRESS_UNCHANGED
  - CiscoConnection.CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION. このオプションでは、コールがリダイレクトの送信先に到着したときに、フィールドが影響を受けます。

詳細については、com.cisco.jtapi.extensions.CiscoConnection のドキュメントを参照してください。

A が B をコールし、B が C にリダイレクトし、C（リダイレクト先）がプロバイダーによって監視されたアドレスを表さない場合、JTAPI により、C について原因コード Ev.CAUSE_NORMAL で CallCtlConnAlertingEv が提供されます。リリース 5.0 よりも前では、このシナリオの原因コードは CiscoCallEv.CAUSE_REDIRECTED を指定していました。

この変更では、C がプロバイダーによって監視されている場合と監視されていない場合の動作の一貫性が維持されています。

（注）　同じシナリオで C が監視されている場合、5.0 よりも前のリリースでも C について CAUSE_NORMAL で CallCtlConnAlertingEv が提供されており、この動作は変更ありません。

ルーティング

JTAPI でのルーティングには、Cisco Unified Communications Manager 上の CTI ルート ポイントの設定が必要です。複数のコールをこのルート ポイントにキューイングできますが、CTI ルート ポイントのデバイス上に設定できるのは、1 本の回線だけです。

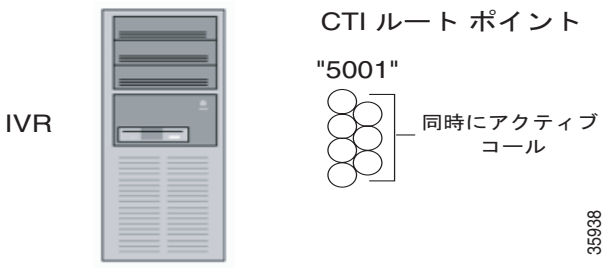
コール センター パッケージで説明されるように、補助ルーティングの JTAPI 実装には、次のアクションが含まれています。

- ルート アドレス上のルート コールバックを登録する
- 多様なルーティング イベント（routeSelect、routeEnd）への応答に対して適切なハンドラを作成する

（注）　CTI ルート ポイントは、同一回線上で同時に無制限の数の着信コールを処理できるデバイスを表します。コールに対しては、javax.telephony.callcenter パッケージのメソッドを使用したルーティング、あるいは javax.telephony.callcontrol パッケージのメソッドを使用した受け入れ、リダイレクト、または接続解除を実行できます。各 CTI ルート ポイントは、Cisco Unified Communications Manager の 1 回線だけで構成できます。1 つの CTI ルート ポイントでサポートされる回線の最大数は 34 です。34 を超える回線をサポートするには、ルート ポイントを追加します。CTI ルート ポイントの構成方法と管理方法の詳細については、『*Cisco Unified Communications Manager Administration Guide*』を参照してください。

図 3-7 に CTI ルート ポイントの構成を示します。

図 3-7　CTI ルート ポイント



Cisco Route Session の実装

コールが RouteAddress で着信されると、この実装では Route Session スレッドが開始され、アプリケーションに RouteEvent が送信されます。次に、このスレッドでは、routeSelect() または endRoute() のどちらかによる RouteEvent に対するアプリケーションの応答時間を計るため、タイマー スレッドが開始されます。アプリケーションが routeSelect (String[] selectedRoutes) で応答した場合、JTAPI ではすべての事前条件が満たされていることを確認してから、配列で指定された最初の送信先へのコールのルーティングが試みられます。送信先が有効かつ利用可能な番号であった場合、コールはルーティングされ、アプリケーションでは RouteUsedEvent と RouteEndEvent が順番に取得されます。送信先が有効かつ利用可能な番号でない場合は、無効、使用中、利用できない送信先などの原因でルーティングにエラーが生じ、アプリケーションでは ReRouteEvent が取得されます。JTAPI では、re-Route Event を送信する前にタイマー スレッドが再開されます。Cisco Unified Communications Manager では、再ルーティングがサポートされないため、ルーティングが失敗した場合、発信者に対してビジー トーンが再生されるか、またはコールが破棄されます。アプリケーションでは、すべての障害インスタンスをクリーン アップしたり、JTAPI に endRoute を送信して RouteSession をクリーン アップできます。アプリケーションが endRoute() に応答しない場合、JTAPI タイマーは再度満了し、JTAPI では、アプリケーションに RouteEndEvent() を送信することで、Route Session がクリーン アップされます。

アプリケーションから selectRoute() メソッドまたは endRoute() メソッドが返される前にルーティング タイマーが満了すると、Cisco Unified Communications Manager によって、登録されていない電話機に対して発呼した場合と同じ処理（ファースト ビジー音の再生）が適用されます。ルート ポイント上に ForwardNoAnswer が設定されている場合、タイマーが満了した時点で、すぐにコールはその番号に転送されます。

コールをルーティングする有効なアドレスを応答できない場合、endRoute とエラーをアプリケーションによって呼び出すこともできます。JTAPI 仕様では、RouteSession インターフェイスで ERROR_RESOURCE_BUSY、ERROR_RESOURCE_OUT_OF_SERVICE、および ERROR_UNKNOWN の 3 種類のエラーが定義されています。endRoute が RouteSession 上で呼び出される場合、この実装では現在、accepts() により RouteAddress でコールが受け入れられ、発信者に対してリングバック音が再生されます。ルート ポイントに対して転送が設定されている場合、Forwarding Timer が満了するとコールが転送されます。

Route Timer の選択

このタイマーは、RouteSelectTimeout=5000 というキーを含む JTAPI.ini コンフィギュレーション ファイルで設定します。単位にはミリ秒を使用します。このタイマーのデフォルト値は、5 秒に指定されていますが、アプリケーションのニーズに応じてタイマーの時間を増減して、Route Session のクリーンアップ効率を改善できます。このタイマーの値は、むやみに大きくしないでください。スレッドとしての各 Route Session は、ルート ポイントへのコールを表すため、これらの Route Session をクリーンアップする必要があります。アプリケーション上でルート イベントの受信と routeSelect/endRoute イベントへの応答の間に深刻な遅延が予測される場合は、アプリケーションによってこのタイマーの時間を適切に延長してください。

Forwarding Timer

現在、システム全体で使用されている Forward on No Answer 用タイマー（つまり、Cisco Unified Communications Manager 上のすべてのデバイスに適用されます）は、Cisco Unified Communications Manager のサービス パラメータ設定により設定できます。このタイマーのデフォルト値は、12 秒に指定されています。今後のリリースには、JTAPI でコールが受け入れられた直後（アプリケーションから endRoute を呼び出したか、またはルーティング タイマーが満了したとき）にルート ポイントに対する転送が機能するように、CTI Route Point 用の別個のタイマーが追加される予定です。

RouteSession の拡張

CiscoRouteSession は、JTAPI 仕様に対する Cisco の拡張の役割を果たします。この拡張の最も重要な点は、基盤となるコール オブジェクトがアプリケーションに公開されることです。CiscoRouteSession.getCall() は CiscoCall を返し、このコールによって、関連するアドレス、Connection などの他のコール モデル オブジェクトが公開されます。この拡張では、アプリケーションに対する追加のエラーも定義されています。

発信者オプションの要約

コールバックがない場合、または routeEvent が RouteSession.routeSelect() または endRoute() で応答されない場合、次の状態になるまで発信者に対して何も送信されません。

- アプリケーション側でルート ポイントの Connection を disconnect() によって解除または reject() によって拒否でき、その結果、発信者に対してビジー トーンが送信される。
- アプリケーション側でコールを受け入れることができ、Forward No Answer が設定されていればこれが作動する。
- アプリケーションによってコールを破棄できる。発信者は受話器を持ったままで、何が起きたかが理解できません。

コールバックを使用すると、アプリケーションによって endRoute() を呼び出した場合、endRoute() が返された後、次の状態になるまで発信者に対してリングバック音が送信されます。

- クライアントがコールを破棄する disconnect() を呼び出す。
- クライアントがコールを redirects() でリダイレクトする。
- scm.ini で設定された Forward on No Answer タイマーが作動し、前の 2 つのオプションがまだ作動していない場合はコールが転送される。
- ルート ポイントに対して転送が設定されていない場合、最初の 2 つのオプションが作動していない場合は発信者に対してリングバック音が送信され続ける。

ルート ポイント使用時の耐障害性

ルート ポイントを使用するアプリケーションで耐障害性を確保するには、2 つの JTAPI アプリケーションを用意し、別個の RouteAddress の登録された 2 つの異なる Cisco Unified Communications Manager にこのアプリケーションを接続する方法があります。たとえば、Application1 では Communications Manager1 を使用して RouteAddress1 を管理します。Application2 では、Communications Manager2 を使用して RouteAddress2 を管理します。Cisco Unified Communications Manager Administration 上では、ルート ポイントが互いにポイントし合うように、これらの CTI ルート ポイントに対する ForwardNoAnswer の設定を管理する必要があります。この例では、RouteAddress1 に FNA=RouteAddess2、RouteAddress2 に FNA=RouteAddress1 を付けます。Communications Manager1 が停止した場合、Application2 が引き継げるようにコールが RouteAddress2 に転送されます。さらに、ProviderShutdown イベントの受信時にそれぞれの適切な Cisco Unified Communications Manager サーバに再接続するように、両方のアプリケーションを設定できます。

冗長性

設定時には、デバイスをデバイス プール内に設定し、静的な Cisco Unified Communications Manager グループに割り当てる必要があります。デバイスは、コール制御のシグナリングを処理する特定の Cisco Unified Communications Manager サーバに登録されます。サーバに障害が起きると、デバイス

はそのグループのバックアップ サーバにフェールオーバーされます。デバイスは、プライマリ サーバがオンライン状態に復旧し、デバイス上のアクティブ コールがなくなるまで待機した後で、プライマリ Cisco Unified Communications Manager サーバにリホームされます。Cisco Unified JTAPI では、バックアップ サーバへの登録中に、一時的なアウト オブ サービスのメッセージを送信することで、アプリケーションにこの移行が通知されます。

クラスタ抽象概念

CTIManager は、クラスタ内にあるすべての Cisco Unified Communications Manager の仮想表現を提供します。Cisco Unified JTAPI アプリケーションは、特定の Cisco Unified Communications Manager ではなく CTIManager と通信します。また、CTIManager によってクラスタ内にある Cisco Unified Communications Manager 間の接続が維持されます。これにより、プロバイダーは CTIManager を使用してクラスタ内のすべてのデバイスを表現できます。図 3-8 は「JTAPI、Cisco Unified Communications Manager、および CTIManager が 1 つのボックス内にある単一ボックス構成」を図示したものです。図 3-9 は「JTAPI を独立したクライアントとして配置した冗長 Cisco Unified Communications Manager と CTIManager」を図示したものです。

クラスタ管理とデバイス プール設定に関する詳細については、Cisco Unified Communications Manager ヘルプの情報を参照してください。

図 3-8　JTAPI、Cisco Unified Communications Manager、および CTIManager が 1 つのボックス内にある単一ボックス構成

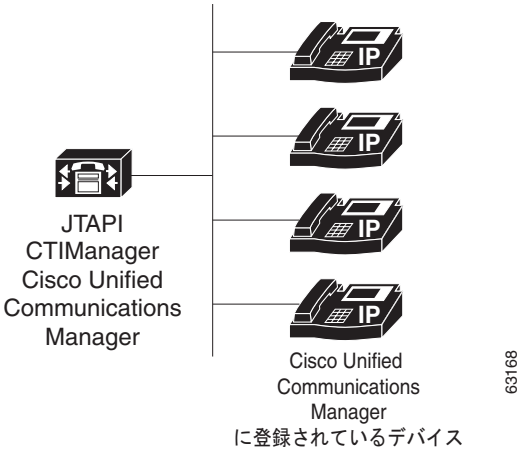
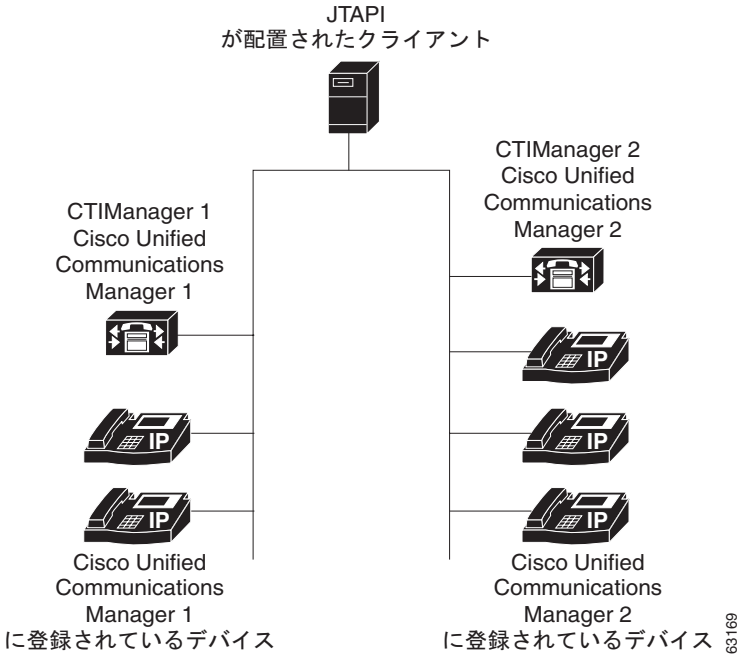


図 3-9　JTAPI を独立したクライアントとして配置した冗長 Cisco Unified Communications Manager と CTIManager



（注）　Cisco Unified Communications Manager の以前のリリースでは、Cisco Unified JTAPI 上で動作するアプリケーションでは、単一の Cisco Unified Communications Manager に登録されたデバイスの制御または監視だけが可能でした。Cisco Unified Communications Manager サーバが停止した場合、Cisco Unified Communications Manager サーバと JTAPI 間の接続が停止し、プロバイダーはシャットダウンします。

Cisco Unified Communications Managerサーバの障害

Cisco Unified Communications Manager に障害が起きた場合、関連デバイスはそのグループにある次の Cisco Unified Communications Manager サーバにリホームされます。このプロセスは、デバイスごとのデバイス プール情報の設定にある Cisco Unified Communications Manager の優先順位付きリストで定義されます。

Cisco Unified Communications Manager サーバの障害時でも、クラスタ内にあるデバイスの部分的な機能停止が生じるだけです。これらのデバイスは、Cisco Unified Communications Manager のフェールオーバーが成功し、セカンダリ Cisco Unified Communications Manager に登録された後、引き続き使用可能になります。

（注）　Cisco Unified IP Phone 7960 などのデバイスでは、デバイス上にアクティブ コールがないときにだけ、セカンダリの Cisco Unified Communications Manager サーバに対するフェールオーバーが行われます。コール中の Cisco Unified Communications Manager サーバの障害では、そのデバイスの観察が停止するだけです。メディア パスはそのまま存在しますが、コール制御は機能しません。

Cisco Unified JTAPI では、CiscoAddrOutOfServiceEv イベントと CiscoTermOutOfServiceEv イベントを使用して、この部分的な機能停止がアプリケーションに伝えられます。Cisco Unified Communications Manager のフェールオーバー時は、JTAPI アプリケーションに対してデバイスが使用

可能になる前に、デバイスは正常にセカンダリ Cisco Unified Communications Manager に登録される必要があります。Cisco Unified JTAPI からは、CiscoAddrInServiceEv イベントと CiscoTermInServiceEv イベントが送信されます。

この間、プロバイダーはイン サービス状態にあります。他の Cisco Unified Communications Manager サーバ上にあるデバイスには、引き続きすべてのコール制御を実行できます。イベントは、個々のアドレスまたは端末のオブザーバ オブジェクトがコールバックされると送信されます。CiscoAddrOutOfServiceEv イベントと CiscoAddrInServiceEv イベントは、AddressObserver を実装しているオブジェクトに送信され、addressChangedEvent() コールバック オブジェクト メソッドを使用してアドレスに追加されます。CiscoTermOutOfServiceEv イベントと CiscoTermInServiceEv イベントは、TerminalObserver インターフェイスを実装しているオブジェクトに送信され、terminalChangedEvent( ) コールバック メソッドを使用して端末に追加されます。

デバイスに現在コールがある場合は、CallObservationEnded メッセージが CallObserver callChangedEvent() のコールバック時に送信され、次に、CiscoAddrOutOfServiceEv メッセージと CiscoTermOutOfServiceEv メッセージが送信されます。

（注） アプリケーションでは、アドレスまたは端末のコール制御関数を呼び出す前に、CiscoAddrOutOfServiceEv、CiscoTermOutOfServiceEv、CiscoAddrInServiceEv、および CiscoTermInServiceEv のイベントの監視および応答を実行する必要があります。このアクションをサポートしない場合、アプリケーション側でシステムの正確な状態を把握できないため、予想外のエラーが生じる可能性があります。

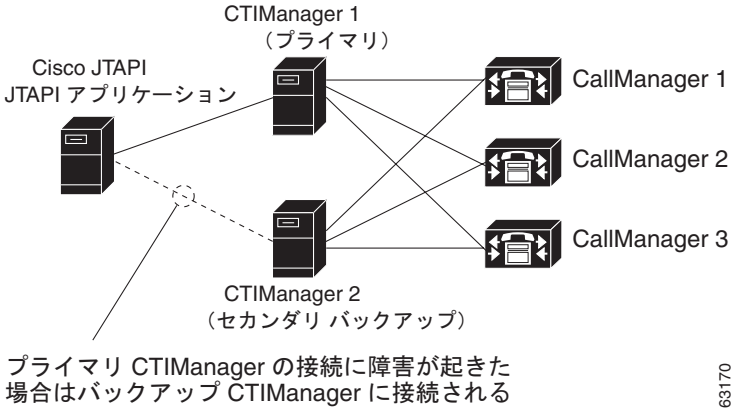
CTI Manager の冗長性

Cisco Unified JTAPI では、CTI Manager 経由での冗長性のための透過型アプリケーションも提供されます。プライマリ CTI Manager に障害が起きると、Cisco Unified JTAPI はバックアップ CTI Manager と自動的に接続され、この再接続がアプリケーションに伝えられます。これにより、1 台の Cisco Unified Communications Manager サーバに接続する代わりに、アプリケーションは CTIManager のセットに接続します。アプリケーションでは、JTAPI を呼び出すときに CTIManager サーバの名前を指定します。

Cisco Unified JTAPI と CTIManager では、両者間の接続切れを検出するために双方向のハートビート信号が維持されます。CTIManager によって、アプリケーションが作動していない状態が検出され、その割り当てられたリソースがクリーン アップされます。図 3-10 はクラスタ内における JTAPI、CTIManager、および Cisco Unified Communications Manager の論理表現を図示したものです。

（注） Cisco Unified JTAPI では、プライマリ CTIManager に正常に接続された後、CTIManager への JTAPI 接続が失敗すると、プライマリまたはバックアップの CTIManager への再接続が交互に試みられます。

図 3-10　クラスタ内における JTAPI、CTIManager、および Cisco Unified Communications Manager の論理表現



CTIManager の冗長性を呼び出す

アプリケーションの起動中に CiscoJtapiPeer の getProvider() メソッドが呼び出されると、Cisco Unified JTAPI では、リストにある最初の CTIManager への接続が試みられ、最初の接続の試みが失敗すると、次の CTIManager への接続が試行されます。リストにあるすべての CTIManager が使用できないか、または接続がすべての CTIManager に拒否された場合、例外がアプリケーションに送信され、それ以上の再接続は試行されません。最初の接続が成功した後、CTIManager または CTIManager への接続に障害が検出されると、Cisco Unified JTAPI では、バックアップまたはプライマリの CTIManager への接続が交互に試みられます。

冗長 CTIManager のリストは、カンマ区切りのリストであり、文字列として CiscoJtapiPeer.getProvider(String providerString) に渡されます。providerString の使用方法は次のとおりです。

- providerString = CTIManager;login=XXX;passwd=YYY;appinfo=ZZZ （非冗長機能）
- providerString = CTIManager1,CTIManager2;login=XXX,passwd=YYY;appinfo=ZZZ （冗長機能）

（注）　appinfo パラメータはオプションなので、アプリケーションでは特定の appinfo パラメータは指定しません。このパラメータは、Cisco Unified JTAPI が JTAPI インスタンス ID とローカル ホスト名から生成します。

また、CiscoJtapiPeer.getServices() メソッドをサポートするために、jtapi.ini ファイル内にさまざまな CTIManager リストを定義できます。Cisco Unified JTAPI では、次の定義を適用できます。

CtiManagers=<CTIManager1>,<CTIManager2>;<CTIManager3>

説明：

<CTIManager1>、<CTIManager2> は、冗長グループを示します。

<CTIManager3> は、非冗長グループを示します。

CTIManager の障害

Cisco Unified JTAPI によって CTIManager との接続の切断が検出されると、アプリケーションにこのサービスの切断が通知されます。該当するオブザーバ上のアプリケーションに次のイベントが送信されます。

- CallObservationEndedEv イベントは、アドレス上にあるすべてのコール オブザーバに送信され、進行中のコールは終了します。これらのコールは物理的には接続されていますが、Cisco Unified JTAPI にはコールの状態変化を送信する機能がないため、アプリケーションによるコールの観察は終了します。
- CiscoAddrOutOfServiceEv イベントが端末上のすべてのアドレスに送信され、CiscoTermOutOfServiceEv イベントが端末に送信されます。
- このプロセスは、プロバイダーのユーザ制御リストにあるすべての端末に対して繰り返されます。（CiscoAddrOutOfServiceEv イベントは、アクティブな AddressObserver があるアドレスにだけ送信され、CiscoTermOutOfServiceEv イベントは、アクティブな TerminalObserver がある端末にだけ送信されます。）
- その後、プロバイダーはアウト オブ サービス状態に設定され、プロバイダー上の ProviderObserver コールバックに ProvOutOfServiceEv イベントが配送されます。

次に、Cisco Unified JTAPI によってリストの次の CTIManager への接続が試みられ、ProvInServiceEv が ProviderObserver に送信されます。あらかじめアプリケーションの制御下で登録されたデバイスは新しい CTIManager に再インストールされます。デバイスが再インストールされると、CiscoAddrInServiceEv イベントと CiscoTermInServiceEv イベントが、それぞれのオブザーバを経由してアプリケーションに送信されます。あらかじめ追加されたすべてのオブザーバは維持されます。デバイス上にコールが存在する場合、コールのスナップショットが、それぞれのコール オブザーバに送信されます。

あらかじめ登録された CTI ポートは同じメディア パラメータで再登録されます。RouteAddress コールバックは以前と同様に維持され、これらのコールは新しい CTIManager 上で復旧されます。ただし、コール スナップショットは RouteAddress に配送されません。

ハートビート

Cisco Unified JTAPI と CTIManager では、CTIManager または JTAPI の障害を検出するためにハートビート信号が維持されます。CTIManager サーバは、双方向のハートビートでハートビート パラメータを制御します。アプリケーションでは、Cisco Unified JTAPI を初期化するときに、希望するサーバ ハートビート インターバルを要求できますが、このインターバルは CTIManager によって上書きされる可能性があります。

アプリケーションでは、jtapi.ini 設定にある DesiredServerHeartbeatInterval を使用して必要なハートビート パラメータを指定します。

Cisco Unified JTAPI では、初期化中にクライアントの要求したハートビート インターバルが指定されます。CTIManager では、クライアント側のハートビート インターバルとサーバ（CTIManager）からハートビートを送信するインターバルが、Cisco Unified JTAPI に対して指定されます。サーバ指定のインターバルの 2 倍の期間にハートビート メッセージが受信されない場合は、クライアントにより接続がティアダウンされます。ハートビートのトラフィックを最小化するには、クライアントからサーバへのすべてのメッセージ、またはサーバからクライアントへのすべてのイベントを、ハートビートの代わりに使用します。

コールの発側の表示名

CiscoCall インターフェイスには、コールの発側と着側の名前を表示できるメソッドが用意されています。アプリケーション上で getCurrentCallingPartyDisplayName() を使用して、発呼側の表示名を取得できます。

JTAPI アプリケーションでは、次のインターフェイスを使用して発側と着側の表示名を取得できます。

```
{
..
```

```
..
    /**
     *This interface returns the display name of the called party in the call.
     *It returns null if display name is unknown.
    */
    public String getCurrentCalledPartyDisplayName();

    /**
     *This interface returns the display name of the calling party.
     *It returns null if display name is unknown.
    */
    public String getCurrentCallingPartyDisplayName();
}
```

表示名は、アドレス オブジェクト内部で保管され、currentCallingAddress と currentCalledAddress が更新されたときに更新されます。コールがアクティブな状態にない場合、またはコールの currentCalling と currentCalled のアドレスが初期化されない場合、NULL が返されます。

（注）　Call.getCurrentCalledAddress() と call.getCurrentCallingAddress() は、電話会議ではサポートされません。また、電話会議では call.getCurrentCalledPartyDisplayName() と call.getCurrentCallingPartyDisplayName() もサポートされません。

SetMessageWaiting

SetMessageWaiting インターフェイスは、アドレスのメッセージ待ちランプまたはインジケータを、アプリケーションで設定するためのメソッドを提供します。このメソッドは、送信先と同じパーティションにあるアドレス上で呼び出します。

次のインターフェイスでは、`destination` で指定したアドレスに対して、メッセージ待ちインジケータを有効または無効にする必要があるかどうかが指定されます。`enable` が true の場合は、メッセージ待機が有効になります（すでに有効になっている場合はそのまま）。`enable` が false の場合は、メッセージ待機が無効になります（すでに無効になっている場合はそのまま）。

```
{
public void setMessageWaiting (java.lang.String destination, boolean enable)
    throws  javax.telephony.MethodNotSupportedException,
            javax.telephony.InvalidStateException,
            javax.telephony.PrivilegeViolationException
}
```

QuietClear

QuietClear は、Cisco Unified Communications Manager の停止、CTIPort を制御しているアプリケーションの停止、または CTIManager の停止などのネットワーク障害が原因で、コールの 2 つの通話者のうち、一方のアドレスがアウト オブ サービスになると、もう一方の端に示されます。この段階では、コールのもう一方の端では、コールの破棄または接続の解除だけが可能です。他の callControl 操作は実行できません。

アウト オブ サービスになった通話者については、アプリケーションには ConnDisconnectedEv や TermConnDroppedEv が送られ、コールのもう一方の端では、ConnFailedEv と CiscoCallEv.CAUSE_TEMPORARYFAILURE の CiscoCause が受信されます。

QuietClear モード時にアプリケーションによって次の機能の起動が試みられると、CiscoJtapiException.CTIERR_OPERATION_FAILER_QUIETCLEAR のエラー コードとともに PlatformException がスローされます。

- コンサルト転送
- コンサルト会議
- ブラインド転送
- 保留
- 保留解除

（注）　アプリケーションは、このモードではコールの破棄だけが可能です。

GetCallInfo

アドレス上の GetCallInfo インターフェイスでは、アドレスに関する CallInfo をアプリケーションによって問い合せることができます。問い合せは、CiscoAddressCallInfo オブジェクトを返します。これには、アクティブまたは保留のコールの数、アクティブまたは保留のコールの最大数、およびこのアドレスでの現行コールのコール オブジェクトに関する情報が含まれます。このインターフェイスは、どんなコールが特定の時刻に特定のアドレスにあるかも示します。

次のインターフェイスを使用して、端末に存在するコールに関する情報を取得します。

```
{ public CiscoAddressCallInfo getAddressCallInfo(Terminal iterminal);
}
```

DeleteCall

DeleteCall インターフェイスでは、アプリケーションが、createCall インターフェイスを使用して作成されたコールを削除できるようにします。このメソッドはコールを受け取り、プロバイダーがイン サービスでない場合、またはコールがアイドル状態ではない場合に InvalidStateException をスローします。DeleteCall により、コールが Invalid 状態に移行されます。

次のインターフェイスが CiscoProvider に追加されます。

```
{ public void deleteCall( Call call ) throws InvalidStateException;
}
```

アプリケーションでは、このインターフェイスを使用して createCall を使用して作成されたコールを削除できます。このメソッドはコールを受け取り、プロバイダーがイン サービスでない場合、またはコールがアイドル状態ではない場合に InvalidStateException をスローします。DeleteCall により、コールが Invalid 状態に移行されます。

コールを正常に削除するには、createCall を使用してアプリケーションでコールを作成し、コールをアイドル状態にする必要があります。

GetGlobalCallID

GetGlobalCallID では、CiscoCallID 上で、コールの nodeID および Global Call ID（GCID）を取得するインターフェイスを提供します。これは内部コール オブジェクトにある GCID 情報を公開します。

次のメソッドが、CiscoCallID インターフェイスに追加されます。

```
{   /**
    * returns the callmanager nodeID of the call
    */
public int getCallManagerID();
```

```
    /**
    * returns the GlobalCallID of the call
    */
public int getGlobalCallID ();
}
```

RTP イベントの GetCallID

GetCallID では、発側や着側などの任意のコール情報にアクセスし、アプリケーション側からコールの RTP イベントにリンクできるように、RTP イベント上のインターフェイスを提供します。

CTIManager からの RTP イベントで受信される callLegID は、クライアント側の ICCNCall の決定に使用されます。このコールは、JTAPI 層に渡され、CiscoCallID の取得元から CiscoCall が判定されます。この情報は、アプリケーションに配送される RTP イベントを構築するために使用されます。

次のインターフェイスが、`CiscoRTPInputStoppedEv`、`CiscoRTPInputStartedEv`、`CiscoRTPOutputStoppedEv`、および `CiscoRTPOutputStartedEv` に追加されます。

```
{ public CiscoCallID getCallID();
}
```

XSI オブジェクト パス スルー

アプリケーションでは、JTAPI と CTI のインターフェイス経由で XML オブジェクトを電話機に渡すことができます。XML オブジェクトには、IP フォン サービス機能から使用可能な電話機上の表示の更新、ソフトキーの更新、有効化、無効化および他のタイプの更新を含めることができます。これにより、電話機への独立した接続を維持することなく、JTAPI および CTI のインターフェイス経由で、アプリケーションから IP フォン サービス機能にアクセスできます。

CiscoTerminal メソッド

アプリケーションでは、CiscoTerminal インターフェイス メソッドを使用して、バイト形式の XSI オブジェクトを Cisco Unified IP Phone に送信できます。このインターフェイスでは、ペイロードが 2000 バイトのデータに制限されます。

CiscoTerminal は CiscoTerminal.REGISTERED 状態であり、そのプロバイダーは Provider.IN_SERVICE 状態である必要があります。正常な応答があった場合は、プッシュされたデータが電話に到着したことを示します。ただし、アプリケーションでは、プッシュ要求による CiscoIPPhoneResponse オブジェクトを含めて、電話から返送される XML を受信することはできない点に注意してください。アプリケーションの要求が成功しなかった場合は、PlatformException がスローされます。データ長が 2000 バイトを超える要求は、すべて拒否されます。

```
public String sendData ( String terminalData ) throws InvalidStateException,
MethodNotSupportedException;
```

この機能をアプリケーションで利用するには、事前に端末に関する TerminalObserver を追加する必要があります。

認証とメカニズム

電話の IP アドレスを必要とする HTTP POST 要求を電話機の Web サーバに送信すると、オブジェクトのプッシュが実行されます。Web サーバは、その要求を解析し、Cisco Unified Communications Manager に返された HTTP により要求を認証して、これを実行し、要求の成功または失敗を示す XML 応答をアプリケーションに返します。

XSI では、Skinny Client Control Protocol（SCCP）によって IP フォン サービス オブジェクトが電話に直接送信されます。JTAPI クライアントは信頼されており、電話の IP アドレスを必要としないので、電話では要求の認証は行われません。実際の XML 内容の詳細については、『*Cisco IP Phone Services Application Development Notes*』を参照してください。

Cisco VG248 および ATA 186 アナログ電話ゲートウェイ

Cisco Unified JTAPI は、Cisco VG248 および ATA 186 アナログ電話ゲートウェイに接続されているアナログ電話の制御をサポートします。Cisco VG248 および ATA 186 アナログ電話ゲートウェイをユーザ制御リストに追加すると、これらのデバイスをアプリケーションで制御できます。

アプリケーションには、他の IP フォンと同様の方法でデバイスに関するイベントが送信されます。また、アプリケーションでは、コールを発信できるほか、API による応答要求を除くその他の機能を呼び出すこともできます。発信が実行されるのは、デバイスが物理的にオフフック状態にある場合だけです。

デバイスに対する API からのコールにアプリケーションが応答できません。アプリケーションが VG248 および ATA 186 端末に対する TerminalConnection で () を応答しようとすると、CiscoJtapiException.COMMAND_NOT_IMPLEMENTED_ON_DEVICE というエラーを示す PlatformException が送出されます。コールに応答するには、受話器を手で持ち上げる必要があります。それによって、API による転送、会議、ブラインド転送、パークなど、その他のコール制御機能を呼び出せるようになります。

DN あたりの複数コール

DN あたりの複数コールは、1 回線（DN）上での複数コールと、これらのコール機能の操作をサポートする機能を表します。Cisco Unified Communications Manager リリース 4.0(1) よりも前では、最大 2 つのコールしかサポートされていませんでした。Cisco JTAPI は、1 回線あたり複数のコールをサポートします。これにより、同一回線上で複数のコールを扱い、この回線上で機能を操作することが可能になります。

DN あたりの複数コールによる、インターフェイスやメッセージ フローの変更はありません。

共用回線のサポート

共用回線は、複数端末での同じ DN のアピアランスを表します。CiscoJtapi は、共用回線をサポートします。共用回線では、同じ DN を持つ他の共用回線端末上で別のコールがアクティブになっていても、共有 DN 端末の制御、特定の共有 DN 端末にあるコールの保留と別の共有 DN 端末からの保留解除、2 本の共用回線間での発呼、特定の共用回線端末からのコール開始を、アプリケーション側から実行できます。

共用回線には、次のインターフェイスがあります。

- CiscoAddress.getInServiceAddrTerminals()：アドレスがイン サービス状態にある端末の配列を返します。

  ```
  Terminal {} getInServiceAddrTerminals();
  ```

- CiscoAddrOutOfService.getTerminal()：アウト オブ サービスの端末を返します。

  ```
  Terminal getTerminal();
  ```

- CiscoAddrInService.getTerminal()：イン サービスの端末を返します。

  ```
  Terminal  getTerminal();
  ```

- CiscoConnection.setRequestController(TerminalConnection tc)：Connection に関連付けられた terminalConnection をアプリケーション側で選択して、パーク、リダイレクト、または接続解除操作を実行できるようにします。SharedLine のシナリオで複数の TerminalConnection がアクティブになっている場合に、これが必要になります。
- CiscoConnection.getRequestController()：アプリケーションが要求コントローラとして設定する TerminalConnection を返します。
- CiscoAddrAddedToTerminalEv：次の条件が発生すると送信されます。
  - SharedDN を含む user controlList に端末またはデバイスが追加され、アプリケーションにイベントが送信された。つまり、ユーザのアドレスが制御リストにあり、制御リストの同じアドレスに新しいデバイスを追加した場合、このイベントが送信されます。
  - EM（エクステンション モビリティ）ユーザが、SharedDN を含むプロファイルを使用して端末にログインした。この場合、このイベントは、既存のアドレスに新しい端末が追加されたことを通知します。
  - ユーザ制御リスト内のデバイスに新しい SharedDN が追加された。

  getTerminal() インターフェイスは、アドレスに追加される端末を返します。

  getAddress() インターフェイスは、新しい端末が追加されるアドレスを返します。
- CiscoAddrRemoveFromTerminalEv：次の条件が発生すると送信されます。
  - SharedDN を含む user controlList から端末またはデバイスが削除された。つまり、ユーザの制御リスト内の共有アドレスを使用したデバイスが制御リストから削除された場合、このイベントが送信されます。
  - SharedDN を含むプロファイルが使用されている端末から、EM（エクステンション モビリティ）ユーザがログアウトした。このイベントは、端末のうち 1 台が既存のアドレスから削除されたことをアプリケーションに通知します。
  - ユーザ制御リスト内のデバイスから新しい SharedDN（SharedLine）が削除された。

  getTerminal() インターフェイスは、アドレスから削除される端末を返します。

  getAddress() インターフェイスは、端末が削除されるアドレスを返します。

SharedLine の変更された動作または新しい動作を次に示します。

- CiscoAddress イベントの動作には、次の変更点があります。
  - JTAPI アプリケーションは、共用回線アドレスの複数の CiscoAddrInServiceEv を受信します。アプリケーションは、CiscoAddrInServiceEv.getTerminal() を使用して、アドレスがイン サービスの端末を取得できます。
  - JTAPI アプリケーションは、共用回線アドレスの複数の CiscoAddrOutOfServiceEv を受信します。アプリケーションは、CiscoAddrInServiceEv.getTerminal() を使用して、アドレスがアウト オブ サービスの端末を取得できます。
  - 最初の共用回線がイン サービスになると、アドレスの状態がイン サービスになります（最初の CiscoAddressInServiceEv が受信されたときなど）。
  - 最後の共用回線がアウト オブ サービスになると、アドレスの状態がアウト オブ サービスになります（最後の CiscoAddressOutOfServiceEv が受信されたときなど）。
- 着信コールの場合、共用回線のすべての回線アピアランスが鳴ります。これは、アプリケーションに対しては、1 つのアクティブ コール（callActiveEv）、1 つの Connection（ConnCreatedEv）、および複数の terminalConnection（共用回線あたり 1 件の TermConnCreatedEv）として示されます。

- コールは、すべての端末に対して示されます。コールが呼び出し状態になっている場合、TerminalConnection の状態は呼び出し中です。共用回線が応答すると、terminalConnection は Active 状態に、共用回線上にあるその他の terminalConnection は Passive 状態になり、その時点のすべての共用回線の callControlTerminalConnection は Bridged 状態になります。コールが保留にされると、すべての TerminalConnection は Active 状態に、callControllTerminalConnection は HELD 状態になります。この時点では、どの端末でもコールを取得できます。取得する側の端末では、terminalConnection が Active 状態のまま callControlTerminalConnection が Talking 状態になり、その他のすべての共用端末の terminalConnection は Passive 状態になります。同時に、CallControlTerminalConnection が HELD 状態から Bridged 状態に変わります。
- ある共用回線から、同じ DN を持つ別の共用回線に発呼できます。この場合、コールには 1 つの Connection と複数の TerminalConnection が含まれます。
- ある共用回線から同じ DN を持つ別の共用回線に発呼する場合、事後条件は 1 つの Connection だけになります。
- 2 つの terminalConnection がアクティブになっている共用回線 Connection（割込みなど）の場合、Connection.Disconnect() を実行しても接続解除されません。
- Passive 状態または Bridged 状態の TerminalConnection が 1 つしかない 1 つの SharedDN Connection だけをアプリケーションが監視している場合、この Connection に対してどんな API を呼び出しても PreConditionException が発生します。
- 前述のシナリオと同様に、アプリケーションが監視しているコールのすべての Connection に Passive 状態または Bridged 状態の TerminalConnection しかない場合、このコールに対するすべての API が PreConditionException をスローします（`Call.Drop()` など）。
- 共用回線上に複数の Active 状態の TerminalConnection がある場合、次のシナリオにおいて Call.drop() は CallInValid を返しません。
  - A が A' との SharedLine で、A' がコールに割り込んでいる、A と B の 2 者間コールの場合。

    アプリケーションは A' および B を監視していません。アプリケーションが Call.drop() を発行した場合、A’ の TerminalConnection は Passive 状態になりますが、コールは無効になりません。
  - 前述同様に、A、A'、A"、B が電話会議を行っている場合。

    アプリケーションは A および A' だけを監視するので、Call.drop() を実行してもコールは無効になりません。A および A' の TerminalConnection だけがパッシブになります。
  - A、A'、B、B' が SharedLine アドレスである場合。

    A が B にコールし、B が応答し、A' と B' がコールに割り込みます。アプリケーションは A および B だけを監視します。この場合、Call.drop() により A および B の TerminalConnection は Passive 状態になりますが、コールは無効になりません。
- TerminalConnection が Passive 状態または Bridged 状態、あるいは Passive/InUse 状態にある場合、TerminalConnection() ですべての API が PreConditionException をスローします。Passive または Bridged 状態では、TerminalConnection で API 端末の ConnectionJoin()（割り込みと呼ばれる）だけを使用できます。TerminalConnection は、現在 TerminalConnection Join() をサポートしていません。
- Connection 上に Active 状態または Talking 状態の複数の TerminalConnection がある場合、アプリケーションは一方を終了してから、この Connection に対して Redirect()、Park()、Disconnect() などの API を発行する必要がある可能性があります。Connection.setRequestController（TerminalConnection tc）API を使用して、TerminalConnection を選択できます。
- SharedLine 端末上のコールが保留され、アプリケーションが Connection.Disconnect () を発行した場合、アプリケーションは Connection.setRequestController（TerminalConnection tc）API を使用して特定の TerminalConnection を設定できます。requestcontroller が設定されていない場合、すべての HeldTerminalConnection が破棄され、Connection が解除された状態になります。1 つの

HeldConnection だけが破棄された場合、その他の SharedLines 端末上にコールが残ります。破棄された端末からはコール アピアランスがなくなるため、このコールに対して、この端末が割り込んだり、機能の操作に参加したりすることはできなくなります。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。共用回線のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

転送と直接転送

転送機能は、コールの転送機能を提供します。

直接転送機能は、回線上にある任意の 2 つのコールを転送する機能を示します。コールのコントローラがドロップし、その他の 2 者がアクティブ状態のままコールに残ることができます。この機能では、1 つの拡張機能がサポートされています。この機能は、コールがどの状態にあっても実行でき、新しい CTI イベントに合わせて再設計することもできます。転送機能には、次の拡張機能があります。

- アプリケーションから、保留中の 2 つのコールを転送する。
- アプリケーションが、OneHeld および OneConnected のコールをあらゆる順序で保持できる。
- アプリケーションから、回線上にある任意の 2 つのコールを転送する。

転送および直接転送の変更されたインターフェイスまたは新しいインターフェイスを次に示します。

- CiscoTransferStarted.getTransferControllers() : SharedLine 用の新しいインターフェイスです。SharedLine が TransferController の場合に、複数の terminalConnection をサポートします。transferController が SharedLine ではないときは、1 つの TerminalConnection だけがリストに含まれます。転送コントローラが監視されていない場合、このメソッドは null を返します。
- CiscoTransferStarted.getTransferController() : 現行インターフェイスであり、通常の転送では現状どおりに動作しますが、SharedLine では動作が異なる場合があります。transferController が SharedLine であるときには、複数の TerminalConnection があります。このメソッドは ACTIVE TerminalConnection を返します。ただし、アプリケーションが ACTIVE TerminalConnection を監視していない場合、このメソッドは PASSIVE TerminalConnection のうち 1 つを返します。
- CiscoTransferEnded isSuccess() : CiscoTransferEnded イベント用の新しいインターフェイスです。転送処理が正常に終了した場合は true を返し、転送に失敗した場合は false を返します。転送は次のイベントにより失敗する可能性があります。
  - CallProcessing が転送を完了する前に、一方がコールをドロップした。
  - CallProcessing が転送の完了を行えない。

JTAPI 転送の変更された動作または新しい動作を次に示します。

- 任意転送において、保留メッセージまたは保留解除メッセージが発生しません。
- 転送要求の事前条件が変更されている場合、コールがどの状態にあってもアプリケーションが転送を発行できます。
- アプリケーションに引数としてアクティブな TerminalConnection が渡されていない場合、Call.consult() は PreConditionException/InvalidArgumentException をスローします。
- コントローラにアクティブな TerminalConnection がない場合、Call.Transfer() は PreconditionException/InvalidArgumentException をスローします。

転送と直接転送のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

会議と参加

会議機能は、1 つのコールで 3 者以上の会議を行う機能を提供します。新しい CTI イベントをサポートするために、CTI 層のイベントが変更され、Cisco Unified JTAPI が拡張されています。

参加機能は、複数のコールを 1 つの電話会議に参加させる機能を提供します。この機能は複数のコールをサポートするようになりました。アプリケーションは、会議に参加するコールの配列を渡す必要があります。

会議と、1 つの電話会議への複数コールの参加処理用に、次の新しいインターフェイスまたは変更されたインターフェイスがあります。

- 次のインターフェイスを使用すると、1 つの電話会議に複数のコールの参加を行うことができます。

  ```
  Call.Conference(Call[] otherCalls)
  ```

  （注）　事前条件として、他のすべての otherCalls がコントローラをコールのレッグとして持つ必要があります。

- 次の新しいまたは変更されたインターフェイスが、CiscoConferenceStartEv にあります。
  - TerminalConnection getHeldConferenceController() : このインターフェイスは 2 つのコールの任意会議だけに有効で、保留されたコールのうち 1 つだけを返します。
  - TerminalConnection[] getHeldConferenceControllers() : このインターフェイスは、複数のコールを参加させる際にすべての保留中のコールを取得します。
  - TerminalConnection getTalkingConferenceController() : このインターフェイスは、会話中の会議コントローラを返します。ただし、会議に参加しているすべてのコールが保留され、会話中の会議コントローラがない場合、このインターフェイスは null を返します。
  - Call getConferencedCall() : このインターフェイスは、会議に参加する多数のコールのうち 1 つだけを返します。3 つ以上のコールがある場合の参加会議に関しては、無意味である可能性があります。
- CiscoConferenceEnded イベントの新しいインターフェイス、isSuccess() :

  このインターフェイスは、会議 が正常に実行されたか失敗したかに応じて、True または False を返します。アプリケーションは、このインターフェイスを使用して、会議が正常に実行されたかどうかを確認できます。次のイベントは会議の失敗として定義されます。
  - アプリケーションが Call.conference(otherCalls[]) 要求を発行し、1 つ以上のコールが会議に参加できなかった場合、この会議は失敗と見なされます。アプリケーションは、getFailedCalls() インターフェイスを使用して、失敗したコールを見つけることができます。
  - 会議ブリッジがなく、会議がまったく開催できなかった場合、アプリケーションは、getFailedCalls() を使用して、会議に参加できなかったコールのリストを取得できます。
  - 会議が開催できる前に会議に参加していた側がドロップした。
- CiscoConferenceEnded イベントのインターフェイス（Call[] getFailedCalls()）は、会議が失敗した際に、会議への参加に失敗したすべてのコールを取得します。

会議の新しい動作または変更された動作を次に示します。

- 任意会議が発生した時点でアプリケーションが確認できるような保留中または保留解除のメッセージはありません。
- 任意会議で、いずれかのコールが通話中状態にある必要があるという事前条件はなくなりました。ただし、他のすべての otherCalls がコントローラをコールのレッグとして持つ必要があります。

- アプリケーションは、2 つ以上の保留中のコールを電話会議に追加できます。finalCall では、コントローラが自動的に Talking 状態に戻ります。
- Call.Conference(otherCalls) 要求にはアクティブ コールを必ず含めてください。会議要求にアクティブ コールが含まれていない場合、要求が失敗します。
- コントローラにアクティブ コールがない場合、Call.Conference(otherCalls) 要求は正常に実行されます。ただし、アクティブ コールが 1 つある場合は、要求に含める必要があります。
- アプリケーションに引数としてアクティブな TerminalConnection が渡されていない場合、Call.consult() は PreConditionException/InvalidArgumentException をスローします。
- コントローラにアクティブな TerminalConnection がない場合、Call.Conference()/Call.Conference(Call[]) は PreconditionException/InvalidArgumentException をスローします。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。会議と参加のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

割り込みとプライバシー イベント通知

割り込み機能は、別端末のアドレスで確立されたコールに共用アドレスが割り込み処理できる機能です。この機能は、TerminalConnection アドレスが Passive 状態で、CallCtlTerminalConnection が Bridged 状態である場合に有効です。Cisco Unified JTAPI のこのバージョンでは、アプリケーション制御による端末（IP フォン）から手動で機能を有効にすることだけがサポートされます。このリリースでは、API から機能を有効にすることはできません。

プライバシー機能は、他の共有アドレスからコールへの割り込み処理を有効または無効にできる機能です。プライバシーを有効にすると、他の共有アドレスからコールに割り込めません。また、無効にすると、割り込みができます。プライバシーは、端末のプロパティを示します。IP フォンには「プライバシー」ソフトキーがあり、このソフトキーを押すと、プライバシーが有効または無効になります。端末のアクティブ コールの場合、プライバシーを動的に有効または無効にすることができます。コールのプライバシーがオンになっているときは、共有アドレス上にあるコール アピアランスの TerminalConnection は「InUse」状態です。CallProgress 中にプライバシー ステータスが変化した場合、CiscoTermConnPrivacyChangedEvent がアプリケーションに配信されます。

Cisco Unified Communications Manager には 2 種類の割り込み機能があります。「割り込み」と呼ばれる組み込みの会議ブリッジと、「C 割り込み」と呼ばれる共用会議ブリッジ リソースが使用されます。アプリケーションにとっては、割り込みと C 割り込みの間でのインターフェイスの変更はありません。ただし、一部動作の違いがあり、これらは付録 A「メッセージ シーケンスの図」にあるメッセージ フローの図に記載されています。

割り込み、C 割り込みおよびプライバシーには次のインターフェイスがあります。

Interface CiscoTerminalConnection.getPrivacyStatus()

```
booleangetPrivacyStatus()
```

このインターフェイスは、端末上にあるコールのプライバシー ステータスを返します。

Interface CiscoTermConnPrivacyChangedEv

```
javax.telephony.TerminalConnectiongetTerminalConnection()
```

CiscoCall.CAUSE_BARGE という新しい原因コードが、割り込みの CiscoCall に追加されています。

割り込みの結果、SharedLine TerminalConnection または CallCtiTerminalConnection が Active 状態または Talking 状態になると、JTAPI は CiscoCall.CAUSE_BARGE として CallCtiCause を提供します。この原因コードは、割り込み処理中に作成される一時コールをドロップする際に使用される CallCtiEvent でも提供されます。

C 割り込みでは、この原因コードは提供されません。

これらのインターフェイスの詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。割り込み、C 割り込み、およびプライバシーのメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

CallSelect と UnSelect のイベント通知

直接転送または参加などの操作を行うために、電話のコールを選択または選択解除できます。SharedLine ユーザがコールを選択すると、回線を共用している他の TerminalConnection は、Passive 状態になり、CallCtlTermiCallConnection は InUse 状態になります。コールの選択が解除されると、CallCtlTerminalConnection が Bridged 状態になります。Passive または InUse の TerminalConnection については、アプリケーションから API を呼び出せません。（転送/会議などの）機能動作中の CallProcessing でも、Select/UnSelect 操作を実行できます。アプリケーションが RemoteInUse 端末を監視している場合、アプリケーションは次のイベントも受け取ります。

たとえば、A と A' が SharedLine の場合、A がコールを選択すると、A' の CallCtlTerminalConnection は Passive 状態または InUse 状態になります。A がコールの UnSelect を実行した場合、A' の CallCtlTerminalConneciton は Passive 状態または Bridged 状態になります。

CallSelect または UnSelect に関するメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

動的 CTIPort 登録

この機能により、コールごとに、またはメディアが確立されるたびに、アプリケーションから IP アドレス（ipAddress）およびポート番号（portNumber）を提供することが可能になります。この機能を使用するには、メディア機能を提示して、アプリケーションでメディア端末を登録する必要があります。このメディア端末でコールが応答されると、アプリケーションに CiscoMediaOpenLogicalChannelEv が送信されます。このイベントは、メディアが確立されるたびに送信されます。アプリケーションはこのイベントに応答し、メディアが確立される IP アドレスとポート番号を示す必要があります。

CiscoMediaTerminal は、アプリケーションによる RTP メディア ストリームの終端を可能にする、特殊な CiscoTerminal を表します。CiscoMediaTerminal は、通常の CiscoTerminal と異なり、物理的なテレフォニー エンドポイントではないため、サードパーティ方式で監視、制御することができます。CiscoMediaTerminal は論理的なテレフォニー エンドポイントを表し、メディアを終端するアプリケーションに関連付けることができます。そのようなアプリケーションには、音声メッセージング システム、Interactive Voice Response（IVR; 対話式音声自動応答）、ソフトフォンなどが含まれます。

（注）　Cisco Unified JTAPI により CiscoMediaTerminal として表現されるのは、CTIPort だけです。

メディアの終端プロセスには 2 つの段階があります。アプリケーションはまず、特定の端末向けのメディアを終端するために、Terminal.addObserver メソッドを使用して、CiscoTerminalObserver インターフェイスを実装するオブザーバを追加します。次に、CiscoMediaTerminal.register メソッドを使用して、その IP アドレス、およびその端末向けの着信 RTP ストリームの送信先となるポート番号を登録します。

コールごとに動的に ipAddress および portNumber を登録するには、アプリケーションは、サポートしている機能だけを提示して登録する必要があります。アプリケーションは、メディアが確立されるたびに送信される CiscoMediaOpenLogicalChannelEv に応答する必要があります。アプリケーションが CiscoMediaOpenLogicalChannelEv に応答する前に何らかの機能が実行された場合、機能が失敗する可能性があります。

Cisco Unified Communications Manager Administration ウィンドウにある Media Exchange Timer に指定された時間内にアプリケーションがこのイベントに応答しない場合、コールが失敗する可能性があります。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。コールごとの CTIPort 動的登録のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

（注） ChangeRTPDefaults インターフェイスは CiscoMediaTerminal ではサポートされていません。

コールごとの CTIPort 動的登録には、次の新しいインターフェイスまたは変更されたインターフェイスがあります。

Interface CiscoMediaOpenLogicalChannelEv Extends CiscoTermEv

int　**getpacketSize**()

遠端のパケット サイズを返します（ミリ秒単位）。

int　**getPayLoadType**()

次の定数の 1 つの遠端ペイロード形式を返します。

CiscoRTPHandle　**getCiscoRTPHandle()**

アプリケーションが setRTPParams 要求を呼び出す必要のある CiscoTerminalConnection オブジェクトを返します。

Interface CiscoRTPHandle

int　**getHandle()**

このオブジェクト（現在は Cisco Unified Communications Manager CallLeg ID）の整数表現を返します。

CiscoProvider

CiscoCall　**getCall (CiscoRTPHandle rtpHandle)**

特定の端末に関連付けられた rtpHandle を持つコール オブジェクトを返します。アプリケーションが CallOpenLogicalChannelEv で CiscoRTPHandle を受信した時点で、コール オブサーバが端末に追加されない場合は、CiscoCall が null である可能性があります。

ルート ポイントでのメディア終端

この機能により、コールごとに、またはメディアが確立されるたびに、IP アドレスおよびポート番号を指定することで、ルート ポイントおよびアプリケーションにあるすべてのアクティブ コールを終端することが可能になります。
この機能を使用するには、メディア機能を提示して、アプリケーションでルート ポイントを登録する必要があります。このルート ポイントでコールが応答されると、アプリケーションに CiscoMediaOpenLogicalChannelEv が送信されます。このイベントは、メディアが確立されるたびに送信されます。アプリケーションはこのイベントに応答し、メディアの終端先となる IP アドレスとポート番号を示す必要があります。

CiscoRouteTerminal は、アプリケーションによる RTP メディア ストリームの終端を可能にする、特殊な CiscoTerminal を表します。CiscoRouteTerminal は、通常の CiscoTerminal と異なり、物理的なテレフォニー エンドポイントではないため、サードパーティ方式で監視、制御することができます。CiscoRouteTerminal は論理的なテレフォニー エンドポイントを表し、コールをルーティングしメディアを終端する必要があるアプリケーションに関連付けることができます。CiscoMediaTerminal とは異なり、CiscoRouteTerminal では複数のアクティブ コールを同時に保持できます。通常、CiscoRouteTerminal は、エージェントが次の発信者に応対できるようになるまでの間、コールをキューに格納するために使用されます。

（注） JTAPI により CiscoRouteTerminal として表現されるのは、RoutePoint 端末だけです。

メディアの終端プロセスには 3 つの段階があります。

ステップ 1　アプリケーションが、CiscoRouteTerminal.register メソッドを使用して、メディア機能をこの端末に登録します。

ステップ 2　CiscoTerminalObserver インターフェイスを実装するオブザーバを、アプリケーションが Terminal.addObserver メソッドを使用して追加します。

ステップ 3　CiscoRTPHandle を使用してプロバイダーから CiscoCall オブジェクトを受信するために、アプリケーションは、CiscoRouteTerminal または CiscoRouteAddress に addCallObserver を追加する必要があります。

アプリケーションは、コールごとに CiscoMediaOpenLogicalChannelEv を受信するので、CiscoRouteTerminal の setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。

CiscoJtapiClient 1.4(x) リリース以前で作成されたすべてのアプリケーションは、メディア終端を必要としない場合、修正して CiscoRouteTerminal.NO_MEDIA_TERMINATION に登録する必要があります。

登録されるメディア機能および registrationType が同じであれば、複数のアプリケーションを同じルート ポイントに登録できます。CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION に登録され、端末オブザーバが追加されたすべてのアプリケーションは、CiscoMediaOpenLogicalChannelEv を受信します。ただし、1 つのアプリケーションだけが setRTPParams を呼び出すことができます。

（注） メディアを終端するアプリケーションは、CallControl パッケージを使用して、コールの応答およびリダイレクトを行う必要があります。コールのルーティングだけを行うアプリケーションは、ルーティング パッケージを使用できます。

（注） アプリケーションが CiscoMediaOpenLogicalChannelEv に応答する前に何らかの機能が実行された場合、機能が失敗する可能性があることが、アプリケーション側で認識されている必要があります。Cisco Unified Communications Manager Administration ウィンドウにある Media Exchange Timeout パラメータに指定された時間内にアプリケーションがこのイベントに応答しない場合、コールが失敗する可能性があります。

ルート ポイントでのメディア終端用に、次の新しいインターフェイスまたは変更されたインターフェイスがあります

Interface CiscoRouteTerminal Extends CiscoTerminal

boolean **isRegistered()**

CiscoMediaTerminal が登録された場合、このメソッドは true を返します。それ以外の場合は false を指定します。

boolean **isRegisteredByThisApp()**

アプリケーションが正常に登録要求を発行した場合、このメソッドは true を返します。また、アプリケーションがデバイスを登録解除するまでの間、true のままになります。これは、CTIManager の不具合によりデバイスがアウト オブ サービス状態にある場合でも有効です。

void **register** `(CiscoMediaCapability[] capabilities, int registrationType)`

CiscoRouteTerminal は CiscoTerminal.UNREGISTERED 状態で存在する必要があり、プロバイダーは Provider.IN_SERVICE 状態で存在する必要があります。

void **setRTPParams** `(CiscoRTPHandle rtphandle, CiscoRTPParams rtpParams)`

アプリケーションは、ipAddress および RTP ポート番号を設定して、コールのメディア ストリームを動的に行います。

void **Unregister()**

CiscoRouteTerminal が登録されており、プロバイダーが Provider.IN_SERVICE 状態であることを確認してください。

Interface CiscoMediaOpenLogicalChannelEv Extends CiscoTermEv

int **getpacketSize**()

遠端のパケット サイズを返します（ミリ秒単位）。

int **getPayLoadType**()

次の定数の 1 つの遠端ペイロード形式を返します。

CiscoRTPHandle **getCiscoRTPHandle()**

アプリケーションが setRTPParams 要求を呼び出す必要のある CiscoTerminalConnection オブジェクトを返します。

Interface CiscoRTPHandle

int　**getHandle()**

このオブジェクト（現在は Cisco Unified Communications Manager CallLeg ID）の整数表現を返します。

CiscoProvider

CiscoCall　**getCall (CiscoRTPHandle rtpHandle)**

特定の端末に関連付けられた rtpHandle を持つコール オブジェクトを返します。アプリケーションが CallOpenLogicalChannelEv で CiscoRTPHandle を受信した時点で、コール オブサーバが端末に追加されない場合は、CiscoCall が null を登録する可能性があります。

これらのインターフェイスの詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。ルート ポイントでのメディア終端のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

リダイレクト時の元の着信者 ID のセット

Cisco Unified JTAPI アプリケーションは、優先される元の着信側 DN をリダイレクト要求内に指定できます。リダイレクト時の元の着信者 ID のセット機能により、別の宛先への Connection 上にあるコールをリダイレクトすることに加えて、アプリケーションが OriginalCalledID に任意の値を設定できます。これにより、アプリケーションは、コールを別のボイス メールへ直接転送できます。たとえば、A が B にコールし、B がコールを C Voice Mail へ転送する場合、アプリケーションは拡張リダイレクト要求内で、優先される元の着信者に C を、また C Voice Mail プロファイルに宛先を指定できます。この要求を使用すると、Cisco Unified Communications Manager の originalCalledParty フィールドに C が指定されたコールが C Voice Mail プロファイルに表示されます。一般的なボイスメール アプリケーションは、originalCalledParty 情報を調べてユーザのボイス メールボックスを特定します。

この機能は、元の着信者を変更してコールを特定の通話者へリダイレクトするあらゆるアプリケーションで使用できます。

（注）　この機能では、lastRedirectedAddress も、リダイレクト要求に指定された preferredOriginalCalledParty に変更されます。

次の callControlConnection インターフェイスが、リダイレクト時の元の着信者 ID のセットに適用されます。

Interface CiscoConnection Extends callControlConnection With Additional Cisco Unified Communications Manager-Specific Capabilities

`javax.telephony.Connection`　**redirect (java.lang.String destinationAddress, int mode, int callingSearchSpace, java.lang.String preferredOriginalCalledParty)**

このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。

インターフェイスの詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。リダイレクト時の元の着信者 ID のセットのメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

シングル ステップ転送

このインターフェイスを使用すると、アプリケーションが特定のアドレスにコールを転送できます。Cisco Unified JTAPI は、JTAPI 1.2 仕様の定義に従って引き続きこのインターフェイスをサポートします。ただし、アプリケーションに配信されるイベントが Cisco Unified JTAPI の以前のバージョンから変更されています。

Cisco Unified JTAPI の以前のバージョンでは、このインターフェイスが使用されると、元のコールは保留状態になり、転送コントローラと宛先の間で新しいコールが作成されました。転送が正常に終了すると、転送コントローラ上の両方のコールが IDLE 状態になります。転送に失敗した場合、元のコールは HELD 状態のままになり、アプリケーションがコールを取得します。転送処理の開始時および完了時に、CiscoTransferStart および終了イベントがアプリケーションに配信されます。

アプリケーションには次のような変更があります。

- 新規コールが作成されない。
- CiscoTransferStartEv および CiscoTransferEndEv がアプリケーションに送信されない。
- 転送操作が失敗した場合も元のコールの状態が維持される。

このインターフェイスの事前条件および事後条件に変更はありません。

シングル ステップ転送に関するメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

API の自動アップデート

Cisco Unified Communications Manager を上位バージョンにアップグレードした場合、Cisco Unified Communications Manager の新しいバージョンとは API の互換性がないことに注意する必要があります。アプリケーションが期待どおりに動作できるように、API を互換バージョンにアップグレードしていることを確認してください。API はクライアント サーバにローカルでインストールされるため、複数のマシンをアップグレードする必要があります。クライアント アプリケーションの数が少ない場合は、Cisco Unified Communications Manager Administration に接続し、Cisco Unified Communications Manager 互換プラグインのダウンロードとインストールを行うことで、簡単に実施できます。

クライアント アプリケーションが複数ある場合、この機能は、アプリケーションが始動時に HTTP 要求を使用して自身の ID を Web サーバに提示し、必要な JTAPI API のバージョンに関する応答を受信するための機能を提供します。

アプリケーションのクラスパスにあるローカルのバージョンと、サーバから取得できるバージョンが比較され、アップグレードが必要かどうかが確認されます。これにより、Cisco Unified Communications Manager に合わせてアプリケーションが jtapi.jar コンポーネントをリフレッシュでき、アプリケーションが自動アップデート可能な jtapi.jar を集中展開することができます。

この機能の実行に必要な API は、updater.jar 形式でパッケージ化されています。jtapi.jar および updater.jar のパッケージには、バージョンの比較に使用できる標準マニフェストが含まれています。

（注）　この機能では、JTAPI Preferences、JTAPITestTools、Updater.jar、javadoc コンポーネントはアップデートされません。アプリケーションにこれらのコンポーネントが必要な場合は、Cisco Unified Communications Manager プラグイン ページから JTAPI をインストールしてください。自動アップデートは、JTAPI リリース 2.0 以降をサポートしています。

詳細については、第 4 章「Cisco Unified JTAPI のインストール」を参照してください。

API の自動アップデートには、次の新しいインターフェイスまたは変更されたインターフェイスがあります。

Class com.cisco.services.updater.ComponentUpdater

Component　**queryLocalComponentVersion** (java.lang.String componentName, java.lang.String path)

IOException、IllegalArgumentException をスローします。

Component　**queryServerComponentVersion** (java.lang.String componentName, java.lang.String urlString)

IOException、IllegalArgumentException をスローし、HTTP クエリーをサーバに送信して、リモート サーバにインストールされているコンポーネントのバージョンを判別します。

Interface com.cisco.services.updater.Component

int　**compareTo** (Component otherComponent)

Component　**fetchFromServer**()

HTTP によりコンポーネントをサーバから取得し、temp.jar というファイル名を付けてローカル ファイル システム上のローカル ディレクトリに書き込みます。

java.lang.String　**getBuildDescription ()**

「a.b(c.d) Release」形式の文字列「Release」にあたるバージョンを返します。

int　**getBuildNumber** ()

a.b(c.d) 形式の「d」にあたるバージョンを返します。

java.lang.String　**getLocation ()**

コンポーネントの位置（文字列形式）。

int　**getMajorVersion** ()

a.b(c.d) 形式のバージョン「a」にあたるバージョンを返します。

int　　**getMinorVersion** ()

a.b(c.d) 形式のバージョン「b」にあたるバージョンを返します。

java.lang.String　　**getName ()**

コンポーネント名を返します。

int　　**getRevisionNumber** ()

a.b(c.d) 形式の「c」にあたるバージョンを返します。

また、JTAPI の自動アップデータ機能により、アプリケーションは Cisco Unified Communications Manager から JTAPI.JAR の最新バージョンを直接ダウンロードできます。

1. アップデータにより、アプリケーションの現在のフォルダに newjtapi.jar ファイルが作成されます。これは、Cisco Unified Communications Manager からダウンロードされた新しいバージョンの jar ファイルを表します。
2. アップデータにより、指定されたクラスパスの component.temp というファイルに現在の jtapi.jar がコピーされます。
3. アップデータにより、現在の jtapi.jar が新しい jtapi.jar ファイルに置換されます。

この処理が終わると、現在の jar ファイルは component.temp になり、新しい jar ファイルは jtapi.jar になります。この処理は、Linux と Windows の両方でサポートされています。

自動アップデータの使用例

```
Command Line : java com.cisco.services.updater.ComponentUpdater <server> <component name>
<login> <passwd>
    Component localComponent, downloadedComponent;
    ComponentUpdater updater = new ComponentUpdater();
    String localPath = updater.getLocalComponentPath(args[1]);
    localComponent = updater.queryLocalComponentVersion("jtapi.jar",localPath);
    localComponent.copyTo("component.temp");
    String provString = args[0] + ";login=" + args[2] + ";passwd=" + args[3];

    CiscoJtapiPeer peer = (CiscoJtapiPeer) (JtapiPeerFactory.getJtapiPeer(null));
    CiscoJtapiProperties tempProp = ( (CiscoJtapiPeerImpl) (peer)). getJtapiProperties();
    tempProp.setLightWeightProvider(true);

    Provider provider = peer.getProvider(provString);
    String url = ( (CiscoProvider) (provider)).getJTAPIURL(); provider.shutdown();
    Component serverComponent = updater.queryServerComponentVersion("jtapi.jar", url);

    downloadedComponent = serverComponent.fetchFromServer();
    int retVal = downloadedComponent.replaces(localComponent);
```

「replaces」API は既存の JTAPI バージョンを新しいバージョンに置き換えます。

（注） アップデータは JTAPI.JAR ファイルだけをアップデートし、JTAPI プラグインにバンドルされている他のサンプル アプリケーションや Cisco JTAPI のドキュメンテーションはアップデートしません。これらの別コンポーネントを入手するには、アプリケーションがプラグインを Cisco Unified Communications Manager からダウンロードしてインストールする必要があります。

デバイス タイプ名の処理に関する変更

現在、TSP ではデバイス タイプ名がデバイス タイプに応じてハードコーディングされています。新しいデバイス タイプが追加された場合は、新しいデバイス タイプ名をサポート対象デバイスのリストに手動で追加する必要があります。CTI はデバイス タイプ名を CTI 自身のキャッシュに取得して格納することがないため、TSP ではこの情報を CTI から取得できません。TSP では、手動による操作がないまま新しいデバイス タイプが追加されたときに、デバイス タイプ名を更新する必要があります。

CTI から送られ、deviceInfo 構造体に格納されているデバイス タイプ名の受信が処理されるように QBE インターフェイスの変更が JTAPI に加えられました。deviceInfo は JTAPI ではまったく使用されておらず、アプリケーションにも公開されません。QBE インターフェイスだけが次のように変更されました。

```
public DeviceRegisteredEvent ( String deviceName, int deviceType, boolean
allowsRegistration, int deviceID, boolean loginAllowed, UnicodeString userID, boolean
controlled, int reasonInt, int registrationType, int unicodeEnabled,int locale,

// added for deviceTypeName change
String devTypeName) {

public DeviceUnregisteredEvent ( String deviceName, int deviceType, boolean
allowsRegistration, int deviceID, UnicodeString userID, boolean controllableBool, int
reasonInt , int locale,
//added for devtypename support
String devTypeName) {
```

CiscoTerminal Filter と ButtonPressedEvents

JTAPI 2.0 リリースよりも前では、Cisco Unified JTAPI アプリケーションは端末イベントを直接制御できませんでした。端末オブザーバに適切なフィルタを設定することで、アプリケーションがボタン押下イベントを受信できるようになりました。アプリケーションで RTP イベントを取得するために、コール オブザーバを追加する必要はなくなりました。

CiscoTermEvFilter を使用して setButtonPressedEv が有効にされている場合、電話機で数字ボタンが押されたときに、アプリケーションは CiscoTermButtonPressedEv を受信します。

CiscoTerminal Filter と ButtonPressedEvent には、次の新しいインターフェイスまたは変更されたインターフェイスがあります

CiscoTerminal

void　　**setFilter (CiscoTermEvFilter terminalEvFilter)**

TerminalObserver に配信されるイベントをアプリケーションがより詳細に制御できるようにします。

CiscoTermEvFilter

boolean　**getButtonPressedEnabled()**

端末のボタン押下イベントに関して、有効または無効のステータスを取得します。デフォルトでは、無効に指定されています。

boolean　**getDeviceDataEnabled()**

端末のデバイス データ イベントに関して、有効または無効のステータスを取得します。デフォルトでは、無効に指定されています。

boolean　**getRTPEventsEnabled()**

端末の RTP イベントに関して、有効または無効のステータスを取得します。デフォルトでは、無効に指定されています。

void　**setButtonPressedEnabled** (boolean enabled)

端末のボタン押下イベントを有効または無効に設定します。

void　**setDeviceDataEnabled** (boolean enabled)

端末のデバイス データ ステータス イベントを Enable または Disable に設定します。

void　**setRTPEventsEnabled (boolean enabled)**

端末の RTP イベントを有効または無効に設定します。

CiscoTermButtonPressedEv

int　**getButtonPressed** ()

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。CiscoTerminal Filter と ButtonPressedEvent のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

発信側番号の変更

この機能は、ルート ポイントから SelectRoute API の発信者 DN を変更することを可能にします。アプリケーションは、selectRoute API で発信側番号変更の配列を渡すことができます。発信側番号変更の配列の長さは、選択されたルートの長さと等しくすることができます。対応する routeSelected インデックスに発信側番号変更の要素がない場合、または要素が null である場合、このルート選択要素には発信者番号変更が設定されません。

2 種類の新しいインターフェイスである getModifiedCallingAddress () と getModifiedCalledAddress () が、発信者または着信者の変更された番号を返すコール オブジェクトに公開されます。変更がない場合、これらのインターフェイスは、getCurrentCallingAddress () インターフェイスおよび getCurrentCalledAddress () インターフェイスと同じ値を返す可能性があります。アプリケーションは、selectRoute API を使用してルート ポイントの制御と発信者番号の変更を行っているだけである場合、

getModifiedCallingAddress インターフェイスの発信者アドレス変更を取得できない可能性があります。アプリケーションは、なんらかの発信者または着信者を制御している場合、発信者番号の変更後にコール制御イベントを受信すれば、正しい値を取得できます。

新たに getRouteSelectedIndex () インターフェイスが、新しい CiscoRouteUsedEvent クラスで公開されました。このクラスは、RouteUsedEvent の拡張で、選択されたルートのインデックスを提供します。このメソッドにアクセスするには、アプリケーションは RouteUsedEvent から CiscoRouteUsedEvent へキャストする必要があります。

例

```
routeSelected[0] = 133555
routeSelected[1] = 144911
routeSelected[2] = 143911
routeSelected[3] = 5005

modifiedCallingNumber[0] =null
modifiedCallingNumber[1] =9721234567
modifiedCallingNumber[2] =9721234568
modifiedCallingNumber[3] =null
```

ルーティングで routeSelected[0] または routeSelected[3] が選択された場合、発信者番号の変更が適用されない可能性があります。

この機能は、特定のユーザの Cisco Unified Communications Manager Administration にある [発信側番号の変更] チェックボックス（デフォルトでは [いいえ]）を管理者が有効にした後にだけ使用できます。これが設定されていない場合、RouteSession.CAUSE_PARAMETER_NOT_SUPPORTED という原因の RerouteEvent がアプリケーションに送信されます。発信者番号を変更するアプリケーションでは、着信者の表示名が影響されることと、それに続く発信者または着信者の機能の対話が不整合が生じる場合があることに注意する必要があります。

発信者番号の変更には、次の新しいインターフェイスまたは変更されたインターフェイスがあります。

CiscoRouteSession

void　　**selectRoute** (java.lang.String[] routeSelected, int callingSearchSpace, String[] modifiedCallingNumber)

このインターフェイスは、アプリケーションが発信者番号を routeSelected アドレスに変更できるようにします。対応する routeSelected 要素に modifiedCallingNumber 要素がない場合、特定の routeSelected 要素にコールがルーティングされた際に発信者番号が変更されません。

CiscoCall

javax.telephony.Address　**getModifiedCalledAddress ()**

アプリケーションが selectRoute API を使用して発信者を変更した場合、このインターフェイスは変更された着信側アドレスを返します。ただし、アプリケーションが、発信者番号を変更するルート ポイントを制御しているだけである場合、この情報は正確ではない可能性があります。発信者番号の変更が行われない場合は、getCurrentCalledAddress インターフェイスと同様に機能します。これは、通常、発信側番号が変更された後に機能が呼び出された場合に getCurrentCalledAddress とは異なります。

javax.telephony.Address　**getModifiedCallingAddress ()**

アプリケーションが selectRoute API を使用して発信者を変更した場合、このインターフェイスは変更された発信側アドレスを返します。ただし、アプリケーションが、発信者番号を変更するルート ポイントを制御しているだけである場合、この情報は正確ではない可能性があります。発信者番号の変更が行われない場合は、このインターフェイスは getCurrentCallingAddress インターフェイスと同様に機能します。

CiscoRouteUsedEvent

int　**getRouteSelectedIndex ()**

このメソッドは、コールのルーティング先となるルートの配列インデックスを返します。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。発信者番号の変更のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

CTI ポートおよびルート ポイントの AutoAccept のサポート

この機能は、CTIPort およびルート ポイントのアドレスの AutoAccept を有効または無効にする機能をアプリケーションに提供します。アドレスの AutoAccept ステータスが変更されると、Cisco Unified JTAPI はイベントを提供してアプリケーションに変更を通知します。

（注）　ルート ポイントでサポートされる回線の最大数は 34 です。

CiscoAddress オブジェクト内に提供された新しい setAutoAcceptStatus() インターフェイスを使用すると、AutoAccept をオンまたはオフに設定できます。同様に、CiscoAddress オブジェクト内に提供された getAutoAcceptStatus() インターフェイスを使用すると、アプリケーションはアドレスの現在の AutoAccept ステータスを照会できます。

アドレスの AutoAccept ステータスが変更されると、アプリケーションは AddressObserver によって CiscoAddrAutoAcceptStatusChangedEv を取得します。このイベントには、AutoAccept ステータスが変更された端末を返す getTerminal() インターフェイスと、AutoAccept がオンまたはオフであるかを示す整数を返す getAutoAcceptStatus() が含まれます。アドレス オブザーバが追加されていない場合、このイベントは提供されません。

次のインターフェイスは、CTIPort および RoutePoint の AutoAccept をサポートしています。

CiscoAddress

int　**getAutoAcceptStatus** (javax.telephony.Terminal terminal)

Ciscoaddress.getAutoAccept(Terminal iterminal) は、端末のアドレスに関する AutoAccept ステータスを返します。

void　**setAutoAcceptStatus** (int autoAcceptStatus, javax.telephony.Terminal terminal)

これにより、アプリケーションは CiscoMediaTerminal や CiscoRouteTerminal のアドレスについて AutoAccept を有効にできます。

CiscoAddrAutoAcceptStatusChangedEv

Public interface: **CiscoAddrAutoAcceptStatusChangedEv**

Extends com.cisco.jtapi.exension.CiscoAddrEv

CiscoAddrAutoAcceptStatusChangedEv イベントは、端末のアドレスの AutoAccept ステータスが変更されるたびにアプリケーションに送信されます。アドレスに複数の端末がある場合、このイベントは個々の端末のアドレスの AutoAccept ステータスごとに送信されます。

このイベントには、次のインターフェイスがあります。

int　**getAutoAcceptStatus** ()

CiscoAddrAutoAcceptStatusChangedEv.getAutoAcceptStatus は、CiscoAddress.AUTOACCEPT_OFF CiscoAddress.AUTOACCEPT_ON 端末上のアドレスの AutoAccept ステータス値を次のように返します。

com.cisco.jtapi.extensions.CiscoTerminal　**getTerminal** ()

このアドレスの AutoAccept ステータスが変更された端末を返します。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。CTIPort および RoutePoint での AutoAccept のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

CiscoTermRegistrationFailed イベント

CiscoMediaTerminal または CiscoRouteTerminal の登録が非同期で失敗すると、アプリケーションにこのイベントが提供されます。登録が失敗すると、通常、アプリケーションは CiscoRegistrationFailedException を取得します。ただし、登録要求が正常に行われても、CTI によって登録が拒否される可能性もあります。このイベントは、登録要求が正常に行われても、登録が拒否された場合に提供されます。このイベントを受信するには、アプリケーションに TerminalObserver が必要です。このイベントを受信したときに提供されるエラー コードによっては、アプリケーションが新しいパラメータを使用して再登録する必要があります。

次に、返されるエラーと、解決するためにアプリケーションが行う必要のある処置の一覧を示します。

エラー メッセージ　CiscoTermRegistrationFailedEv.MEDIA_CAPABILITY_MISMATCH

説明　端末がすでに登録されているため、登録できない。同じメディア機能で再度登録を実行してください。

推奨処置　同じ機能で再登録してください。

エラー メッセージ　CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_NONE

説明　端末がメディア終端タイプ「none」ですでに登録されているため、登録できない。

推奨処置　メディア終端タイプ「none」で再登録してください。

エラー メッセージ　CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_STATIC

説明　端末が静的メディア終端ですでに登録されているため、登録できない。静的登録の場合、再度登録を実行できません。

推奨処置　端末が UnRegister 処理されるまで待機してください。

エラー メッセージ　CiscoTermRegistrationFailedEv.MEDIA_ALREADY_TERMINATED_DYNAMIC

説明　端末が動的メディア終端ですでに登録されているため、登録できない。

推奨処置　動的メディア終端で再登録してください。

エラー メッセージ　CiscoTermRegistrationFailedEv.OWNER_NOT_ALIVE

説明　端末を登録しようとすると、登録が競合状態になる。

推奨処置　端末を再登録してください。

このイベントには次のインターフェイスが定義されています。

int　　**getErrorCode** ()

この例外の errorCode を返します。

メッセージ フローの変更はありません。

SelectRoute インターフェイスの拡張

SelectRoute インターフェイスが拡張されて、PreferredOriginalCalledNumber パラメータと PreferredOriginalCalledOption パラメータを取るようになりました。これにより、コールがルーティングされるときに、OriginalCalled 値を指定された PreferredOriginalCalledNumber に再設定することが可能になりました。このインターフェイスは、PreferredOriginalCalledNumber、PreferredOriginalCalledOption のリストを取り、それらを RouteSelected リストに対応付けます。RouteSelected リストのインデックス I にある Route にコールがルーティングされる場合、インデックス I の PreferredOriginalCalledNumber と PreferredOriginalCalledOption が使用されます。パラメータのさまざまな値によって、アプリケーションは次のように動作します。

（注） x の下で、コールがルーティングされるインデックスを選択してください。たとえば、コールが Route n へルーティングされる場合の x 値は n に等しくなります。インデックス x の PreferredOriginalCalledOption が無効または範囲外である場合は、JTAPI によってデフォルトの CiscoRouteSession.DONOT_RESET_ORIGINALCALLED に設定されます。また、PreferredOriginalCalledOption が null である場合、すべてのルーティングが CiscoRouteSession.DONOT_RESET_ORIGINALCALLED オプション付きで処理されます。

PreferredOriginalCalledOption[x] が CiscoRouteSession.RESET_ORIGINALCALLED に設定された場合

- RouteSelected リストに Route R1、R2 ..Rn が含まれる場合、preferredOriginalCalled リストに O1、O2、… On が含まれます。R1 が使用可能な場合、R1 へコールがルーティングされて、OriginalCalledNumber が O1 に設定されます。R1 がビジーで R2 が使用可能な場合、R2 へコールがルーティングされて、OriginalCalledNumber が O2 に設定され、以降同様に続きます。
- RouteSelected リストに Route R1、R2 ..Rn が含まれる場合、preferredOriginalCalled リストに O1、O2、… Om が含まれます（m < n）。R1 が使用可能な場合、R1 へコールがルーティングされて、preferredOriginalCalled が O1 に設定されます。R1 がビジーで R2 が使用可能な場合、R2 へコールがルーティングされて、OriginalCalledNumber が O2 に設定され、m になるまで、以降同様に続きます。Route m+1 からは、Rm+1 が使用可能な場合、Rm+1 へコールがルーティングされて、OriginalCalledNumber が Rm+1 に設定され、以降同様に続きます。最終的に、Rn が使用可能な場合、コールが Rn へルーティングされ、OriginalCalledNumber が Rn " に設定されます。
- RouteSelected リストに Route R1、R2 ..Rn で preferredOriginalCalled リストが NULL のとき、R1 が使用可能な場合、R1 へコールがルーティングされて、OriginalCalledNumber が R1 に設定されます。R1 がビジーで R2 が使用可能な場合、R2 へコールがルーティングされて、OriginalCalledNumber が R2 に設定され、以降同様に続きます。

PreferredOriginalCalledOption[x] が CiscoRouteSession.DONOT_RESET_ORIGINALCALLED に設定された場合

- RouteSelected リストに Route R1、R2 ..Rn が含まれる場合、preferredOriginalCalled リストに O1、O2、..On が含まれます。この場合、利用可能なルートの 1 つにコールがルーティングされ、OriginalCalledNumber は変更されません。
- RouteSelected リストに Route R1、R2 ..Rn が含まれる場合、preferredOriginalCalled リストに O1、O2、… Om が含まれます。m < n では、利用可能なルートの 1 つにコールがルーティングされ、OriginalCalledNumber は変更されません。

- RouteSelected リストに Route R1、R2 ..Rn が含まれ、preferredOriginalCalled リストが NULL である場合、利用可能なルートの 1 つにコールがルーティングされ、OriginalCalledNumber は変更されません。

（注）　OriginalCalled が PreferredOriginalCalled に設定されると、LastRedirectingParty 番号も PreferredOriginalCalled に再設定されます。

SelectRoute インターフェイスの拡張には、次の新しいインターフェイスまたは変更されたインターフェイスがあります。

int　**selectRoute** (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] preferredOriginalCalledNumber, int[] preferredOriginalCalledOption)

コールをルーティング可能な 1 つ以上の宛先を選択します。

PreferredOriginalCalledOption には次のいずれかの値を指定できます。

static int　**DONOT REESET_ORIGINALCALLED**

PreferredOriginalCalledOption のオプション パラメータ値。OriginalCalled を再設定しないことを指定します。

static int　**REESET_ORIGINALCALLED**

PreferredOriginalCalledOption のオプション パラメータ値。OriginalCalled を preferredOriginalCalledNumber に再設定します。

インターフェイスの変更点については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。SelectRoute インターフェイスの拡張のメッセージ フローを確認するには、付録 A「メッセージ シーケンスの図」を参照してください。

コールのプレゼンテーション インジケータ

コールのプレゼンテーション インジケータ（PI）は、エンド ユーザに対して Calling/Called/CurrentCalling/CurrentCalled/LastRedirecting の名前および番号を隠したり見せたりする機能をアプリケーションに提供します。JTAPI には、通話者の PI 値を取得する CiscoCall の関数があります。この PI 情報を使用して、通話者の情報をエンド ユーザに提示します。これらの関数は、true または false の値を返します。値が「true」であれば表示が「許可」状態であり、「false」であれば「制限」状態であることを示します。

電話会議では、CiscoCall のこのインターフェイスは正しい値を返しません。アプリケーションは、コール内のすべての Connection を繰り返して確認し、Connection の作成先アドレスに関連付けられた PI 値を取得する必要があります。CiscoConnection で提供されるインターフェイスは、getAddressPI( ) です。

CiscoCall の次の新しいインターフェイスは PI 値を取得します。

CiscoCall

boolean　**getCalledAddressPI()**

getCalledAddressPI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getCallingAddressPI()**

getCallingAddressPI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getCurrentCalledAddressPI()**

CurrentCalledAddressPI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getCurrentCalledDisplayNamePI()**

CurrentCalledDisplayNamePI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getCurrentCallingAddressPI()**

getCurrentCallingAddressPI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getCurrentCallingDisplayNamePI()**

getCurrentCallingDisplayNamePI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

boolean　**getLastRedirectingAddressPI()**

getLastRedirectingAddressPI に関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

CiscoConnection の次のインターフェイスは、Connection に関連付けられたアドレスの PI 値を取得します。

CiscoConnection

boolean　**getAddressPI()**

Connection の作成先アドレスに関連付けられた PI を返します。true が返された場合、アプリケーションはアドレス名を表示します。false が返された場合、アプリケーションはアドレス名を表示しません。

メッセージ フローの変更はありません。

Progress 状態から Disconnect 状態への変換

ヨーロッパの PSTN で、API を通じて未割り当ての電話番号にコールが発信された場合、アプリケーションは、原因コードが CiscoCallEv.CAUSE_UNALLOCATEDNUMBER に設定された ConnFailedEv イベントを受信します。米国の PSTN の場合は、まったくイベントを受信しないことがあります。

欧州の PSTN を経由した場合と米国の PSTN を経由した場合の動作を一貫性のあるものとし、同時に下位互換性の問題にも対処するために、UseProgressAsDisconnectedDuringErrorEnabled という新しいサービス パラメータが JTAPI バージョン 1.4(3.21) で jtapi.ini ファイルに追加されています。このパラメータを有効（1 = 有効、0 = 無効、デフォルトは無効）にすると、両方の場合にアプリケーションが ConnFailedEv を受信します。

デバイス ステート サーバ

デバイス ステート サーバは、端末のすべてのアドレスの累積的な状態情報を提供します。これらのイベントは、端末イベントとして配信されます。これらのイベントを受信するには、TerminalObserver を追加する必要があります。

次の状態があります。

- IDLE：端末のどのアドレスにもコールが存在しない場合、その端末の DeviceState は IDLE と見なされ、Cisco Unified JTAPI からアプリケーションに `CiscoTermDeviceStateIdleEv` が送信されます。
- ACTIVE：端末のいずれかのアドレスに発信コール（CTI の状態が、Dialtone、Dialing、Proceeding、Ringback または Connected）または着信コール（CTI の状態が Connected）が存在する場合、その端末の DeviceState は ACTIVE と見なされ、Cisco Unified JTAPI からアプリケーションに `CiscoTermDeviceStateActiveEv` が送信されます。
- ALERTING：端末のアドレスに発信コール（CTI の状態が Dialtone、Dialing、Proceeding、Ringback、または Connected）または着信コール（CTI の状態が Connected）が存在し、その端末の 1 つ以上のアドレスに未応答の着信コール（CTI の状態が、Offering、Accepted、または Tinging）が存在する場合、その端末の DeviceState は ALERTING と見なされ、Cisco Unified JTAPI からアプリケーションに `CiscoTermDeviceStateAlertingEv` が送信されます。
- HELD：端末のいずれかのアドレスですべてのコールが保留中（CTI の状態が OnHold）の場合、その端末の DeviceState は HELD と見なされ、Cisco Unified JTAPI からアプリケーションに `CiscoTermDeviceStateHeldEv` が送信されます。

新規イベント

- CiscoTermDeviceDeviceStateIdleEv
- CiscoTermDeviceStateActiveEv
- CiscoTermDeviceStateAlertingEv
- CiscoTermDeviceStateHeldEv

新規インターフェイスと変更されたインターフェイス

`public int getDeviceState()` は、端末のデバイス状態を返します。

CiscoTermEvFilter の次の新規インターフェイスは、デバイス状態の設定と取得を行います。

void　**setDevideStateActiveEvFilter**(boolean filterValue)

CiscoTermDeviceStateActiveEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。

void　**setDeviceStateAlertingEvFilter**(boolean filterValue)

CiscoTermDeviceAlertingEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。

void　**setDeviceStateHeldEvFilter**(boolean filterValue)

CiscoTermDeviceHeldEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。

void　**setDeviceStateIdleEvFilter**(boolean filterValue)

CiscoTermDeviceIdleEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。

boolean　**getDeviceStateActiveEvFilter**()

CiscoTermDeviceStateActiveEv フィルタの状態を取得します。

boolean　**getDeviceStateAlertingEvFilter**()

CiscoTermDeviceStateAlertingEv フィルタの状態を取得します。

boolean　**getDeviceStateActiveEvFilter**()

CiscoTermDeviceStateAlertingEv フィルタの状態を取得します。

boolean　**getDeviceStateActiveEvFilter**()

CiscoTermDeviceStateAlertingEv フィルタの状態を取得します。

インターフェイスの変更に関する詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。

Forced Authorization Code と Client Matter Code

Forced Authorization Codes（FAC）は、外線、市外通話、国際通話など、特定のクラスの着信番号（DN）にコールを発信する前に有効な認証コードの入力をユーザに要求します。認証情報は、Cisco Unified Communications Manager CDR データベースに記録されます。

Client Matter Codes（CMC; クライアント マター コード）は、コールを発信する前にユーザがコードを入力できるようにします。クライアント マター コードを使用すると、発信したコールに会計コードや請求コードを割り当てることができます。クライアント マター コードの情報は、Cisco Unified Communications Manager CDR データベースに記録されます。

サポートされるインターフェイス

Cisco Unified JTAPI では、次のインターフェイスで FAC と CMC をサポートしています。

- Call.Connect()
- Call.Consult()
- Call.Transfer(String)
- Connection.redirect()
- RouteSession.selectRoute()

Call.Connect() と Call.Consult()

アプリケーションがこれらのインターフェイスのいずれかを使用して、FAC、CMC、またはその両方を必要とする DN にコールを発信すると、CiscoToneChangedEv が CallObserver に配信されます。このイベントには、その DN でどちらのコードが必要とされているかについての情報も格納されています。このイベントが FAC_CMC 機能によって配信された場合でも、getCiscoCause() インターフェイスは CiscoCallEV.CAUSE_FAC_CMC を返します。getTone() インターフェイスは、ZIPZIP トーンが再生されていることを示す CiscoTone.ZIPZIP を返します。

CiscoToneChangedEv を受信した後、アプリケーションは connection.addToAddress インターフェイスを使用して、適切なコード（末尾に # を付ける）を入力する必要があります。コードの数字と末尾の # は、T302 タイマーで指定されている文字間タイマー値内に 1 文字ずつ入力することも、T302 タイマー値内に全文字を入力することもできます。T302 タイマーの値は、Cisco Unified Communications Manager Administration で設定します。

FAC と CMC の両方が必要な場合

両方のコードを必要とする DN の場合、最初のイベントは常に FAC に適用され、2 番目のイベントは CMC に適用されます。ただし、2 つのコードをシャープ記号（#）で区切ることにより、両方のコードを同じ要求で送信することもできます。最初の要求で送信されるコードによっては、2 番目のイベントが省略されます。

アプリケーションは両方のコードを同時に送信できますが、次の例に示すように、両方のコードの最後に # を付ける必要があります。

```
connection.addToAddress("1234#678#")
```

ここで、1234 は FAC を表し、678 は CMC を示します。

この場合、アプリケーションは 2 番目の CiscoToneChanged を受信しません。

最初の CiscoToneChangesEv に、getWhichCodeRequired()=CiscoToneChanged.FAC_CMC_REQUIRED と getCause()=CiscoCallEv.CAUSE_FAC_CMC の両方が含まれます。

この場合、次のようなケースが考えられます。

- アプリケーションが FAC と CMC を同じ connection.addToAddress(code1#code2#) 要求で送信する。この場合、アプリケーションは 2 番目の CiscoToneChangedEv を受信しません。
- アプリケーションが最初の connection.addToAddress(code#1) 要求で FAC だけを送信する。この場合、アプリケーションは 2 番目の CiscoToneChangedEv で getWhichCodeRequired()=CiscoToneChangedEv.CMC_REQUIRED を受信します。
- アプリケーションが最初のコードの一部だけ、または最初のコードの全部と 2 番目のコードの一部だけを送信する（末尾に # が付いていない場合、そのコードは不完全と見なされ、T302 タイマーの時間が経過すると、システムがコードの検証を試みます）。コードが不完全な場合、アプリケーションは 2 番目の CiscoToneChangedEv で getWhichCodeRequired()=CiscoToneChangedEv.CMC_REQUIRED および getCause()=CiscoCallEv.CAUSE_FAC_CMC を受信します。

PostCondition タイマー

PostCondition タイマーは、connection.addToAddress インターフェイスを呼び出してコードを送信するたびにリセットされます。FAC および CMC の末尾には # を付ける必要があります（たとえば、Connection.addToAddress(「1234#」) のように入力します。この場合、1234 が FAC と見なされます)。すべてのコードが入力されている場合は、T302 タイマーの時間が経過すると、コールが発信されます。すべてのコードが入力されていない場合は、リオーダー音が再生されます。このケースでは、コールが発信された場合でも、アプリケーションが postConditionTimeout を含む PlatformException を受信することがあります。これを回避するには、JTAPI Preferences を使用して、PostCondition のタイムアウト値を大きくする必要があります。

アプリケーションが call.connect() または call.consult() を使用してコールを開始して、PostCondition タイムアウトの制限時間内に Cisco Unified IP Phone から FAC または CMC（# を含む）が入力されなかった場合、実際にはコールが発信されても、postCondition タイムアウトを含む platformException を受信することがあります。これを回避するには、JTAPI Preferences を使用して、PostCondition のタイムアウト値を大きくする必要があります。

共用回線

開始側の通話者が共用回線を使用している場合は、アプリケーションが connection.addToAddress インターフェイスを使用して追加の番号を渡す前に、setRequestController を使用してアクティブな terminalConnection を設定する必要があります。

無効または欠落したコード

T302 タイマーの有効期限が超過する前に入力されたコードが無効な場合またはコードが入力されなかった場合は、コールが拒否されて callCtlCause 原因コードに CiscoCallEv.CAUSE_FAC-CMC が設定されます。

Call.transfer(String) と Connection.redirect()

FAC と CMC をサポートするために、これらのインターフェイスに 2 つの文字列パラメータ（facCode と cmcCode）が追加されました。これらのコードのデフォルト値は、null 値を表します。

コードを必要とする DN に対してこれらの要求が発行された場合、CiscoToneChangedEv は配信されません。FAC、CMC、またはその両方を必要とする DN へ条件的にリダイレクトされるコールは、いずれかのコードが正しくない場合でも拒否されず、接続が維持されます。

RouteSession.selectRoute()

2 つの追加の文字列パラメータの配列（facCode と cmcCode）が FAC と CMC をサポートしています。各 routeselect 要素に DN のコードを指定できます。コードが不要な DN に対しては null 値を指定する必要があります。これらのコードのデフォルト値は、null 値です。

最初の routeselected 要素に適切なコードが含まれていない場合は、配列に含まれる次の要素が試されます。すべての要素が正しくない場合は、reRouteEvent がアプリケーションに送信されます。

（注）　FAC または CMC コードを必要とする DN への転送（Call Forward）はサポートされていません。Address API を使用して転送番号をこのような DN に設定することはできますが、このような番号へのコール転送は拒否されます。

スーパー プロバイダー（デバイス認証の無効）

通常、システム管理者は JTAPI アプリケーション ユーザを設定する際に、特定の端末（Cisco Unified IP Phone およびデバイス）を各アプリケーション ユーザに関連付けます。そのため、アプリケーション ユーザが制御または監視できる端末は、自分に関連付けられている端末に限定されます。スーパープロバイダー機能を使用すると、Cisco Unified Communications Manager クラスタ内のすべての端末をアプリケーションで制御したり監視することができます。

CiscoProvider の新しいインターフェイスである createTerminal() を使用すると、terminalName を指定して端末を作成できます。JTAPI には、インターフェイスを通じて terminalName を取得する機能が用意されていません。CiscoProvider.createTerminal(terminalName) は、端末を返します。端末がプロバイダー ドメインにすでに存在している場合、JTAPI はその既存の端末を返します。

2 つ目の新しいインターフェイスである CiscoProvider.deleteTerminal() を使用すると、CiscoProvider.createTerminal() インターフェイスで作成された CiscoTerminal オブジェクトを削除できます。端末オブジェクトが存在しない場合や、端末が CiscoProvider.createTerminal() インターフェイスで作成されたものでない場合は、JTAPI が例外をスローします。

また、JTAPI では、CiscoProviderCapabilities の新しいインターフェイスとして canObserveAnyTerminal() も提供されています。アプリケーション ユーザがこのインターフェイスを使用できるようにするには、Cisco Unified Communications Manager Administration のユーザ設定を変更します。アプリケーションはこのインターフェイスを使用して、自身に createTerminal(terminalName) インターフェイスを呼び出す能力があるかどうかを判別できます。このインターフェイスを呼び出す能力がないアプリケーションがこのインターフェイスを呼び出した場合は、JTAPI から PrivilegeViolationException がスローされます。Cisco Unified Communications Manager クラスタ内に存在しない terminalName を指定した場合は、JTAPI から InvalidArgumentException がスローされます。

Q.Signaling（QSIG）パス交換

QSIG パス交換は、QSIG トランクを介して接続されている PBX 間でコールが転送されるときに、リアルタイム プロトコル（RTP）パスを最適化するネットワーク機能です。パス交換の実行中は、アプリケーションからの機能要求が無視されて JTAPI からアプリケーションに例外がスローされる短い期間が発生します。

RTP パスが始端および終端の PBX をつなぐダイレクト パスに最適化されると、コール の Global Call ID が変更されます。JTAPI には、このコールをモニタリングするための新しいインターフェイスが用意されています。

ネットワーク イベント

以前のリリースの Cisco Unified JTAPI では、クラスタ外のアドレスへのコールを開始すると、遠端のアドレスに CallCtlConnNetworkReachedEv イベントと CallCtlConnNetworkAlertingEv イベントが配信されていました。

Cisco Unified Communications Manager 4.0 以降では、これらのイベントは配信されません。これらのバージョンでは、遠端アドレスの CallCtlConnection が OFFERED 状態から ESTABLISHED 状態に移行します。アプリケーションは、遠端アドレスの CallCtlConnOfferedEv および CallCtlConnEstablishedEv を受信します。CallCtlConnNetworkReachedEv および CallCtlConnNetworkAlertingEv イベントはアプリケーションに配信されません。ネットワーク イベントを受信するには、ゲートウェイに対して設定されているルート パターンの「オーバーラップ送信を許可」フラグをオンにする必要があります。

これらのイベントの配信をアプリケーションで制御できるようにするために、jtapi.ini の新しいパラメータ AllowNetworkEventsAfterOffered が導入されています。[オーバーラップ送信を許可] フラグをオンに設定できないアプリケーションでは、この jtapi.ini パラメータを使用して発信コールのネットワーク イベントを受信できます。

このパラメータをオンにするには次の手順を実行します。

ステップ 1　jtprefs を実行して、適切なオプションを選択します。Cisco Unified JTAPI がデフォルト ディレクトリにインストールされている場合は、c:¥winnt¥java¥lib に jtapi.ini ファイルが作成されます。jtapi.ini ファイルがすでに存在する場合は、jtprefs を実行しなくてもこのファイルを直接編集できます。

ステップ 2　Add AllowNetworkEventsAfterOffered=1 をファイルの最後に追加して、ファイルを保存します。

ステップ 3　Cisco Unified JTAPI を再インストールするたびに上記の手順を繰り返します。

AllowNetworkEventsAfterOffered フラグが有効になると、アプリケーションは遠端アドレスに対して、CallCtlConnOfferedEv、CallCtlConnNetworkReachedEv または CallCtlConnNetworkAlertingEv、および CallCtlConnEstablishedEv を受信します。

CHAPTER 4

Cisco Unified JTAPI のインストール

この章では、Cisco Unified Communications Manager 6.0 以降のリリース用の Cisco Unified Java Telephony API（JTAPI）クライアント ソフトウェアのインストール方法と設定方法について説明します。

この章は次のトピックで構成されています。

概要

Cisco Java Telephony API（JTAPI）実装は、JTAPI アプリケーションを実行するすべてのクライアント マシン上に存在する Java クラスによって構成されています。これらのアプリケーションが正しく動作するためには、事前に Cisco Unified JTAPI 実装をインストールする必要があります。JTAPI アプリケーションを Cisco Unified Communications Manager マシンで実行するか、別のマシンで実行するか、両方で実行するかにかかわらず、JTAPI アプリケーションを実行するすべてのマシンに必ず Cisco Unified JTAPI のクラスをインストールしてください。

これまで JTAPI クライアントのインストールは Windows プラットフォームだけでサポートされていました。5.0 リリースから、表 4-1 に示す Linux、Windows、および Solaris プラットフォームでの JTAPI クライアントのインストールおよびアンインストール プロセスを統一するために、JTAPIInstaller が提供されています。InstallShield MultiPlatform（ISMP）インストーラを実行すると、インストールに必要なファイル セットが取得された後、Linux 版および Solaris 版の場合はバイナリ（.bin）ファイルが生成され、Windows 版の場合は実行可能ファイル（.exe）が生成されます。

表 4-1　でサポートされている JVM のバージョンCisco Unified Communications Manager

プラットフォーム	リリース	CiscoUnified Communications Manager リリース 4.x	CiscoUnified Communications Manager 5.x、6.x、7.x
Linux	AS 3.0	IBM JVM 1.3.1 IBM JVM 1.4.2 Sun JVM 1.3.1 Sun JVM 1.4.2	Sun JVM 1.5.0.4 Sun JVM 1.4.2
	Red Hat 7.3	IBM JVM 1.3.1 IBM JVM 1.4.2 Sun JVM 1.3.1 Sun JVM 1.4.2	SunJVM 1.5.0.4 Sun JVM 1.4
Solaris	6.2（SPARC）	Sun JVM 1.3.1 Sun JVM 1.4.2	Sun JVM 1.5.0.4 Sun JVM 1.4.2
Windows	9x	Sun JVM 1.3.1 Sun JVM 1.4.2	Sun JVM 1.4.2
	2000 NT 4.0+ XP（32 ビット） 2003	Sun JVM 1.3.1 Sun JVM 1.4.2 Sun JVM 1.5.0_13	Sun JVM 1.5.0.4 Sun JVM 1.4.2
	Vista（32 ビット）	Sun JVM 1.3.1 Sun JVM 1.4.2	Sun JVM 1.5.0.4 Sun JVM 1.4.2

（注）　Cisco Unified Communications Manager 5.0 にアップグレードした場合は、JTAPI アプリケーションがインストールされているアプリケーション サーバまたはクライアント ワークステーション上の JTAPI クライアント ソフトウェアをアップグレードする必要があります。JTAPI クライアントをアップグレードしないと、アプリケーションの初期化に失敗します。アップグレードする必要がある場合は、「*Cisco Unified JTAPI ソフトウェアのインストール*」の説明に従って適切なクライアントをダウンロードしてください。

アップグレード後の JTAPI クライアント ソフトウェアを以前のリリースの Cisco Unified Communications Manager とともに使用できません。

現在の JTAPI バージョンの確認

インストーラから JTAPI のバージョン番号を確認するには、次のコマンドのうちのいずれかを使用します。

- CiscoJtapiVersion.exe - silent -W newversion.check="1" -goto showversion
- CiscoJtapiClient-linux.bin -silent -W newversion.check=="1" -goto showversion
- CiscoJtapiClient-solarisSparc.bin -silent -W newversion.check=="1" -goto showversion
- CiscoJtapiClient-solarisX86.bin -silent -W newversion.check=="1" -goto showversion

これらのコマンドにより、コマンドを実行したフォルダに「jtapiversion.txt」というファイルが作成されます。このファイル内に A.B(C.D) の形式で JTAPI のバージョンが記述されます。

警告　**リリース 4.X またはそれよりも前のリリースの JTAPI SDK が存在する場合、最初にそれらを手動でアンインストールしないと、新しいソフトウェアのインストールが失敗します。**

以前のバージョンの JTAPI がシステムに存在している状態で現行バージョンにアップグレードしようとすると、以前のリリースのアンインストーラが起動します。Windows システムで以前のリリースのアンインストーラをサイレント モードで起動するには、次のコマンドを使用します。

```
CiscoJtapiClient.exe -silent -W newversion.silent="1"
```

JTAPI Preferences のユーザ インターフェイス ユーティリティ ツールがインストールされている場合は、このツールを使用して現在インストールされている JTAPI のバージョンを確認することもできます。

ステップ 1　[スタート] > [プログラム] > [CiscoJTAPI] > [JTAPI Preferences] の順に選択します。次のメニューが表示されます。

ステップ 2　「ReadMe」ファイルを選択します。このファイルに、現在インストールされている Cisco Unified JTAPI のバージョンが記載されています。

Cisco Unified JTAPI ソフトウェアのインストール

Cisco Unified JTAPI ソフトウェアのインストール モードには次の種類があります。

- サイレント インストール呼び出し
- コマンドライン呼び出し
- エンド ユーザ インストール

ISMP インストーラを実行すると、インストール操作のために Java Runtime Environment（JRE; Java ランタイム環境）バージョンが一時的にインストールされます。この JRE はアンインストール操作時に削除されます。

（注）　Cisco Unified JTAPIInstaller での JRE の配布は、Sun Microsystems, Inc. と Cisco Systems, Inc. の間の契約に基づいて行われます。

警告　**リリース 4.X またはそれよりも前のリリースの JTAPI SDK が存在する場合、最初にそれらを手動でアンインストールしないと、新しいソフトウェアのインストールが失敗します。**

サイレント インストール呼び出し

JTAPIInstaller のサイレント インストール呼び出しを使用すると、アプリケーションのインストーラに JTAPIInstaller を組み込むことができます。

JTAPIInstaller をサイレント モードで起動するアプリケーションでは、次のいずれかのコマンドを使用できます。

- Linux プラットフォーム：**CiscoJTAPIClient-linux –silent**
- Solaris（Sparc）プラットフォーム：**CiscoJTAPIClient-solarisSparc.bin –silent**
- Solaris（X86）プラットフォーム：**CiscoJTAPIClient-solarisX86.bin –silent**
- Windows（Win9X、Win ME、Win2K、WinXP）プラットフォーム：**CiscoJTAPIClient.exe –silent**

JTAPIInstaller で新規インストールまたはアップグレード/ダウングレードを実行すると、インストール先フォルダが自動的に検出されて、サイレント インストールが実行されます。ユーザがインストール時に指定した適切なフォルダ、またはデフォルト フォルダ（サイレント インストールの場合）に、JTAPI サンプル アプリケーションと JAR ファイルが格納されます。ただし、以前のバージョンが存在する場合は、JTAPIInstaller がアプリケーションのパスを判断できないため、デフォルト フォルダ、Lib、および JTAPITools が作成されて、それらのフォルダにアプリケーションがインストールされます。

Windows クライアントの場合は、インストール時にレジストリが更新されて新しいインストール情報が書き込まれます。Linux 版の場合は、ユーザのホーム ディレクトリに jtapiver.ini というファイルが作成されます。

コマンドライン呼び出し

コマンドラインから対話形式で JTAPIInstaller を実行するには、コマンド プロンプトから次のいずれかのコマンドを入力します。

- Linux プラットフォーム：**CiscoJTAPIClient-linux.bin –console**
- Solaris（Sparc）プラットフォーム：**CiscoJTAPIClient-solarisSparc.bin –console**
- Solaris（X86）プラットフォーム：**CiscoJTAPIClient-solarisX86.bin –console**
- Windows（Win9X、WinME、Win2K、WinXP）プラットフォーム：**CiscoJTAPIClient.exe -console**

コマンドライン モードは、Linux システムなど、GUI をサポートしていないシステムに JTAPI をインストールする場合に便利です。すべてのインストール手順が文字ベースのメニューで示され、ユーザはインストール時の条件に基づいて一連の入力を行うように求められます。このモードでは GUI ベースのインストーラで提供されているその他のオプションもすべて使用できます。

エンド ユーザ インストール

JTAPI Installer では、エンド ユーザ インターフェイスとして Java Foundation Classes（JFC）Swing インターフェイスが使用されます。インストール手順を開始すると、ユーザは一連の情報の入力と確認を行うように求められます。

インストーラにより、ターゲット システムの「_uninst」フォルダへのアンインストーラの作成とインストールが行われます。このフォルダは JTAPI アプリケーションのパス（通常、Windows の場合は C:¥Program Files¥JTAPITools、Linux の場合は $HOME/.jtapi/bin）内に配置されます。ユーザはこのパスを使用してアンインストーラを起動できます。

インストーラまたはアンインストーラを実行したフォルダには ismpInstall.txt（または ismpUninstall.txt）というログ ファイルが作成され、このファイルにインストール手順についての詳細情報がすべて保存されます。このファイルには製品インストール中に発生した各種イベントのトレースの完全なリストが含まれるため、エラーの確認にも使用できます。

インストール手順

以降のセクションでは、Linux、Solaris、および Windows プラットフォームでのインストール手順について説明します。

Linux および Solaris プラットフォーム

Cisco Unified JTAPI のインストールと JTAPI Preferences のユーザ インターフェイスでは、複数の言語がサポートされています。

Cisco Unified JTAPIInstaller は次の項目をローカル ディスク ドライブにインストールします。

- JTAPI Java クラス（$HOME/.jtapi/lib ディレクトリ）
- JTAPI Preferences（$HOME/.jtapi/bin ディレクトリ）
- JTAPI サンプル アプリケーション makecall、jtrace（$HOME/.jtapi/bin ディレクトリ）
- JTAPI ドキュメント（$HOME/.jtapi/bin/doc ディレクトリ）

Linux または Solaris プラットフォームに Cisco Unified JTAPI ソフトウェアをインストールするには、次の手順に従います。

ステップ 1　Cisco Unified JTAPI クライアント ソフトウェアをインストールするコンピュータにログインします。

ステップ 2　適切な ISMP インストーラを探して起動します。

- CiscoJTAPIClient-linux.bin（Linux OS の場合）
- CiscoJTAPIClient-solarisSparc.bin（Solaris Sparc OS の場合）
- CiscoJTAPIClient-solarisX86.bin（Solaris X86 OS の場合）

ステップ 3　Cisco Unified JTAPI Installer の指示に従います。

（注）　Cisco Unified JTAPI ソフトウェアはデフォルト ドライブにインストールされます。たとえば Linux の場合、デフォルト ディレクトリは $HOME/.jtapi/lib です。

Linux および Solaris でのインストールの検証

JTAPI が正常にインストールされたことを確認するには、次の手順に従います。

ステップ 1　$HOME ディレクトリに .jtapiver.ini ファイルが作成されていることを確認します。

ステップ 2　$HOME/.jtapi/bin フォルダに JTAPI プログラム ファイルとドキュメントが存在することを確認します。
makecall、jtrace、Locale_files、および doc フォルダを探します。

ステップ 3　$HOME/.jtapi/lib に JTAPI ライブラリが存在することを確認します。
jtapi.jar、jtracing.jar、および updater.jar ファイルを探します。

ステップ 4　クラスパスに jtapi.jar が存在することを確認した後、$HOME/.jtapi/bin ./_jvm/bin/java のコマンドライン プロンプトから次のコマンドを実行します。

```
com.cisco.services.jtprefs.jtprefsFrame
```

これにより、[JTAPI Preferences] ダイアログボックスが表示されます。

（注） JTPrefs アプリケーションが存在しない場合は、次のコマンドを入力して jtapi.ini ファイルを生成できます。

```
< jview | java > CiscoJtapiVersion -parms
```

このコマンドにより、現在のディレクトリに jtapi.ini ファイルが生成されます。

ユーザはインストール時に $HOME 以外のフォルダを選択して JTAPI をインストールできます。その場合は、指定したフォルダ内に .jtapi というフォルダが作成され、そのフォルダの中に bin フォルダと lib フォルダが作成されて対応するファイルがコピーされます。たとえば /home/jtapiuser というフォルダを選択した場合のフォルダ構造は次のようになります。

/home/jtapiuser/.jtapi/bin：このフォルダには makecall、jtrace、Locale_files、および doc フォルダが含まれます。

/home/jtapiuser/.jtapi/lib：このフォルダには jtapi.jar、jtracing.jar、および updater.jar ファイルが含まれます。

この場合、ステップ 4 のコマンドは /home/jtapiuser/.jtapi/bin フォルダから実行します。

Windows プラットフォーム

Cisco Unified JTAPI のインストールと JTAPI Preferences のユーザ インターフェイスでは、複数の言語がサポートされています。Cisco Unified JTAPI Installer は次の項目をローカル ディスク ドライブにインストールします。

- JTAPI Java クラス（%SystemRoot%¥java¥lib ディレクトリ）
- JTAPI Preferences（Program Files¥JTAPITools ディレクトリ）
- JTAPI サンプル アプリケーション makecall、jtrace（Program Files¥JTAPITools ディレクトリ）
- JTAPI ドキュメント（Program Files¥JTAPITools¥doc ディレクトリ）

Windows プラットフォームに Cisco Unified JTAPI ソフトウェアをインストールするには、次の手順に従います。

ステップ 1　Cisco Unified JTAPI クライアント ソフトウェアをインストールするコンピュータにログインします。

ステップ 2　すべての Windows プログラムを終了します。

ステップ 3　ISMP インストーラ（CiscoJTAPIClient.exe）を探して起動します。

ステップ 4　インストーラの指示に従います。

（注） Cisco Unified JTAPI ソフトウェアはデフォルト ドライブにインストールされます。たとえば Windows NT の場合、デフォルト ディレクトリは C:¥WINNT¥Java¥lib です。

Windows でのインストールの検証

JTAPI を利用して発呼する makecall アプリケーションを使用すると、Windows に JTAPI が適切にインストールされているかどうかを検証できます。makecall アプリケーションを使用するには、次の手順に従います。

ステップ 1　Windows NT のコマンドラインから、Cisco Unified JTAPI ツールがインストールされているディレクトリに移動します。デフォルトでは、このディレクトリは C:¥Program Files¥JTAPITools です。

ステップ 2　次のコマンドを実行します。

java CiscoJtapiVersion

ステップ 3　次のコマンドを実行します。

java makecall <*server name*> <*login*> <*password*> **1000** <*phone1*> <*phone2*>

（注）　*server name* には、Cisco Unified Communications Manager のホスト名または IP アドレス（たとえば CTISERVER）を指定します。

phone1 変数と *phone2* 変数には、ユーザ設定に従って制御される IP フォンまたは仮想電話のディレクトリ番号を指定します。詳細については、『*Cisco Unified Communications Manager Administration Guide*』を参照してください。

login 変数と *password* 変数には、Cisco Unified Communications Manager の [ユーザの設定] ウィンドウで設定したユーザ ID とパスワードを指定します。

アップグレードの自動インストール

自動インストール機能には、アプリケーションの起動時に HTTP 要求を使用して Cisco Unified Communications Manager Web サーバにそのアプリケーションの API のバージョンを提示し、必要な JTAPI API のバージョンに関する応答を受信する機能があります。アプリケーションのクラスパスにあるローカルのバージョンと、サーバから取得できるバージョンが比較され、アップグレードが必要かどうかが確認されます。

更新後の API をインスタンス化するように初期化プロセスが変更され、サーバにインストールされているコンポーネントが確認されて、必要に応じてそれらのコンポーネントがダウンロードされます。

この機能を使用すると、Cisco Unified Communications Manager に合わせてアプリケーションが jtapi.jar コンポーネントをリフレッシュでき、アプリケーションで自動アップデート可能な jtapi.jar を一元的に展開できます。

この機能の実行に必要な API は、updater.jar 形式でパッケージ化されています。jtapi.jar および updater.jar のパッケージには、バージョンの比較に使用できる標準マニフェストが含まれています。これにより API 書き込みがアップデートから保護されるため、アプリケーションでバージョン クラスをインスタンス化する必要がなくなります。

場所とコンポーネントを指定してこの機能を実行すると、jtapi.jar がサーバからダウンロードされてローカル ディレクトリにコピーされます。アプリケーションでは、ダウンロードした jtapi.jar を既存のバージョンに上書きするか、新しい jtapi.jar にアクセスするようにクラスパスを変更することができます。

（注）　自動インストールでは、JTAPI Preferences、TAPITestTools、updater.jar、javadoc コンポーネントはアップデートされません。アプリケーションにこれらのコンポーネントが必要な場合は、Cisco Unified Communications Manager プラグイン ウィンドウから JTAPI をインストールしてください。

Cisco Unified JTAPI Preferences の設定

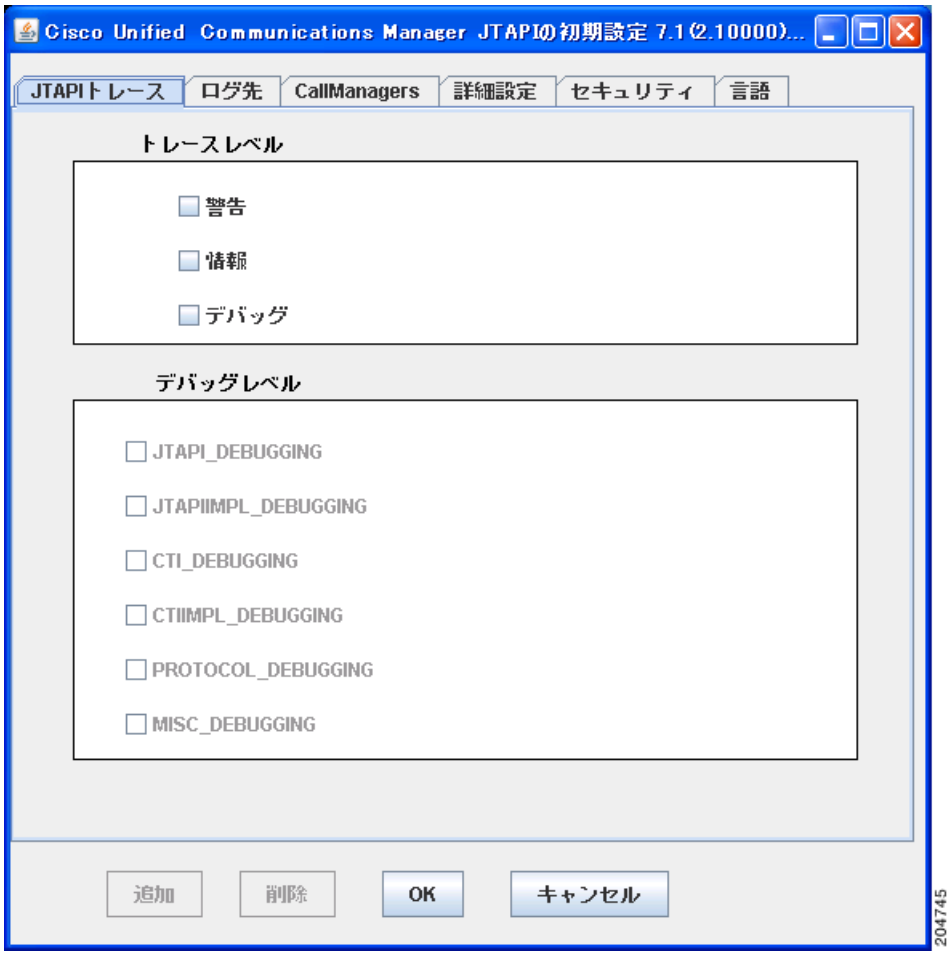
トレース レベル、トレースの保存先、およびその他のシステム パラメータを設定するには、Cisco Unified JTAPI Preferences アプリケーション（JTPREFS）を使用します。デフォルトでは、Cisco Unified JTAPI Preferences は Program Files¥JTAPITools ディレクトリにインストールされます。Cisco Unified JTAPI Preferences ユーティリティを開くには、[スタート] > [プログラム] > [Cisco Unified JTAPI] > [JTAPI Preferences] の順に選択してください。

このセクションでは、Cisco Unified JTAPI Preferences アプリケーションの使用方法について説明します。このセクションの内容は次のとおりです。

[JTAPI トレース] タブ

[JTAPI トレース] タブでは JTAPI レイヤのトレース設定だけを変更できます。[セキュリティ] タブ でセキュリティ インストールのトレース レベルを有効にできます。図 4-1 に、Cisco Unified JTAPI Preferences アプリケーションの [JTAPI トレース] タブを示します。ウィンドウ タイトルには JTAPI のバージョン番号が表示されます。

図 4-1　[JTAPI トレース] タブ



[JTAPI トレース] タブでは、表 4-2 に示す JTAPI トレース レベルを有効または無効にできます。

表 4-2　JTAPI トレース レベル

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
トレース レベル				
警告 WARNING	なし	該当なし	該当なし	低レベルの警告イベント。
情報 INFORMATIONAL	なし	該当なし	該当なし	ステータス イベント。
デバック DEBUG	なし	該当なし	該当なし	最高レベルのデバッグ イベント。
デバッグ レベル				
JTAPI_DEBUGGING	なし	該当なし	該当なし	JTAPI のメソッドおよびイベントのトレース。
JTAPIIMPL_DEBUGGING	なし	該当なし	該当なし	内部 JTAPI 実装トレース。
CTI_DEBUGGING	なし	該当なし	該当なし	JTAPI に送信された Cisco Unified Communications Manager イベントのトレース。
CTIIMPL_DEBUGGING	なし	該当なし	該当なし	内部 CTICLIENT 実装トレース。
PROTOCOL_DEBUGGING	なし	該当なし	該当なし	CTI プロトコルの完全なデコード。
MISC_DEBUGGING	なし	該当なし	該当なし	各種の低レベル デバッグ トレース。

[ログ先] タブ

[ログ先] タブでは、JTAPI によるトレースの作成方法とトレースの保存方法を設定できます。図 4-2 に、Cisco Unified JTAPI Preferences アプリケーションの [ログ先] タブを示します。表 4-3 に、ログ先のフィールドに関する説明を示します。

図 4-2　　[ログ先] タブ

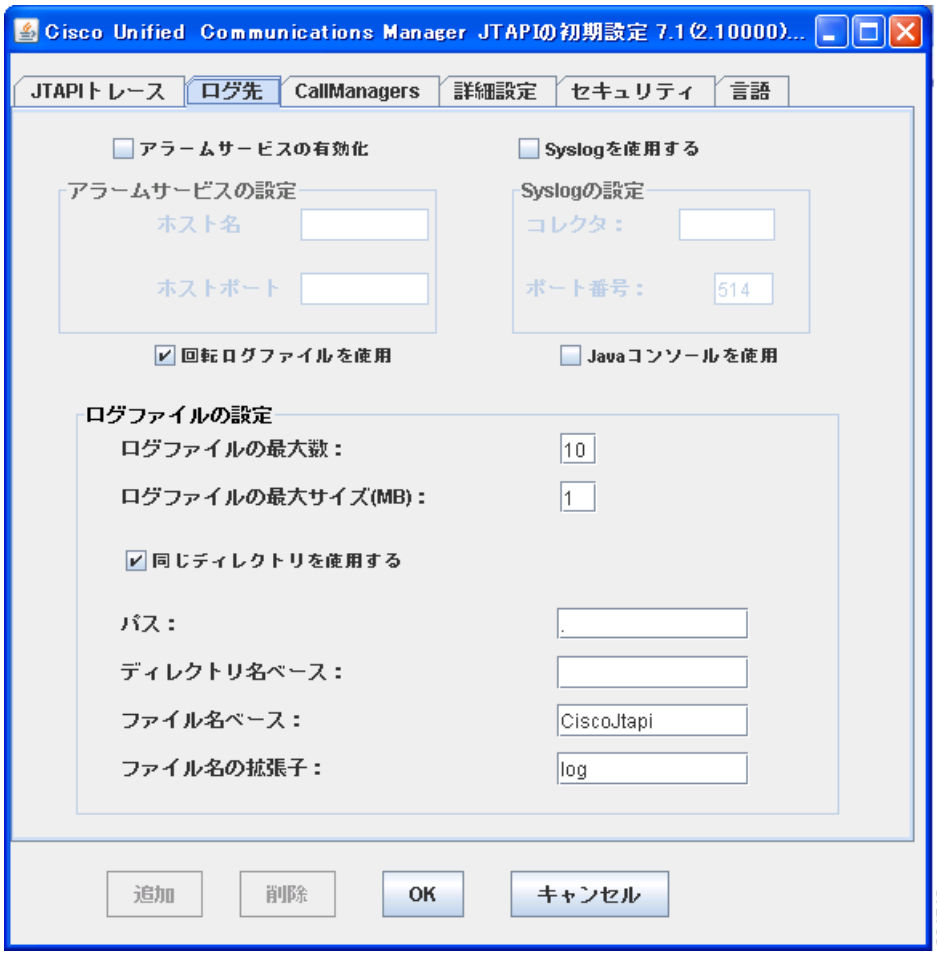


表 4-3　　[ログ先] フィールド

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
アラームサービスの有効化 Enable Alarm Service （UseAlarmService）	0	NA	NA	このオプションを有効にすると、指定したマシンで実行されているアラーム サービスに JTAPI アラームが送信されます。このオプションを有効にする場合は、ホスト名とポート番号を指定する必要があります。

表 4-3　[ログ先] フィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
Syslog を使用する Use Syslog （UseSyslog）	FALSE	NA	NA	このオプションを有効にすると、[コレクタ（Collector）] フィールドと [ポート番号（Port Number）] フィールドで指定された UDP ポートにトレースが送信されます。トレースは syslog コレクタ サービスによって収集され、CiscoWorks2000 サーバに送信されます。
アラームサービスの設定（Alarm Service Settings）				
ホスト名 Host Name		NA	NA	このフィールドを使用して、アラーム サービス サーバのホスト名を指定します。
ホストポート Host Port		NA	NA	このフィールドを使用して、アラーム サービス サーバのホスト ポートを指定します。
Syslog の設定（Syslog Settings）				
コレクタ Collector	0	NA	NA	このフィールドを使用して、トレースを収集する Syslog コレクタ サービスを指定します。
ポート番号 Port Number	514	NA	NA	このフィールドを使用して、コレクタの UDP ポートを指定します。
回転ログファイルを使用 Use Rotating Log Files （SyslogCollector）	FALSE	NA	NA	このフィールドを使用して特定のパスおよびフォルダにトレースを送信できます。作成できるログ ファイルの数は 2 ～ 99 個です。ログ ファイルは番号順に書き込まれ、最後のファイルがいっぱいになると最初のファイルに戻ります。ログ ファイルのサイズは 1 MB ずつ増加します。
Java コンソールを使用 Use Java Console （UseSystemDotOut）	FALSE	NA	NA	このオプションを有効にすると、標準出力またはコンソール（コマンド）ウィンドウにトレースが送信されます。
Log File Settings				
ログファイルの最大数 Maximum Number of Log Files （NumTraceFiles）	10	2	1000	書き込むログ ファイルの最大数を指定します。
ログファイルの最大サイズ（MB） Maximum Log File Size （TraceFileSize）	1048576	1048576	NP	書き込むログ ファイルの最大サイズを指定します。

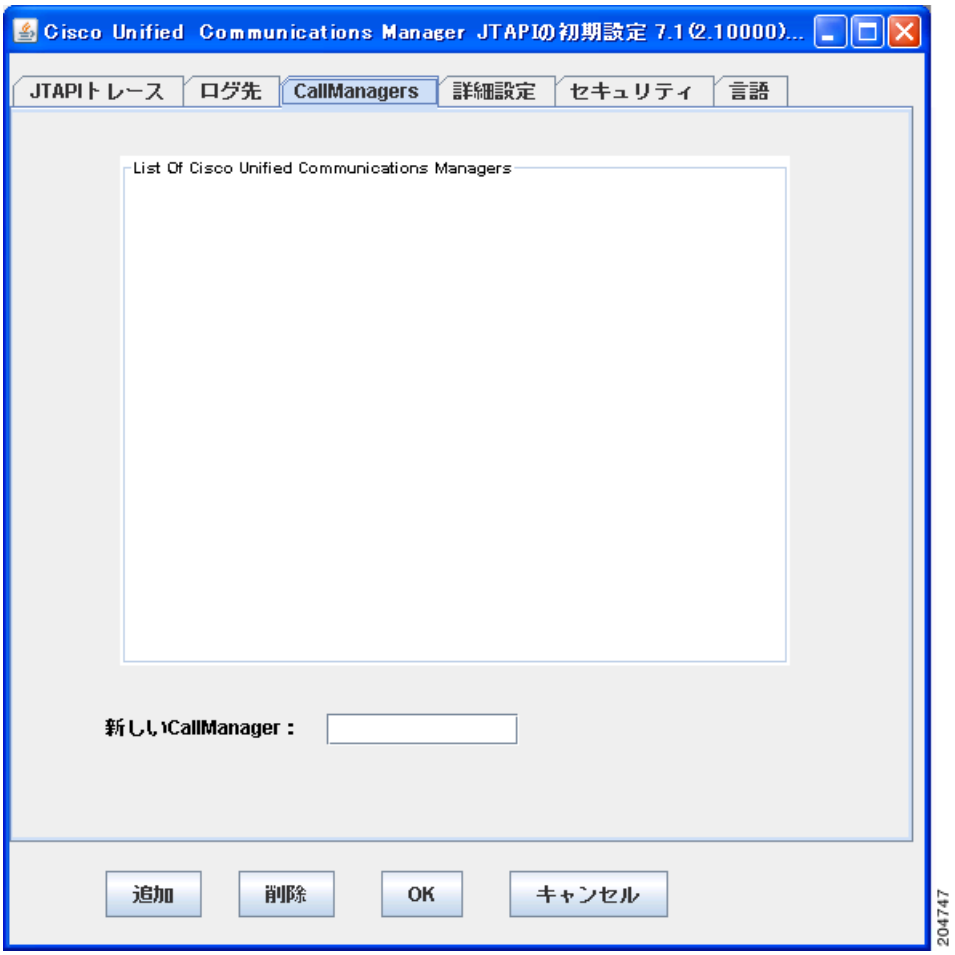
表 4-3　[ログ先] フィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
同じディレクトリを使用する Use the Same Directory（UseSameDirectory）	1	NA	NA	アプリケーションの各インスタンスで同じフォルダ名を使用するかどうかを指定します。 このオプションを有効にすると、同じディレクトリにログ ファイルがトレースされます。この場合は、JTAPI アプリケーションのインスタンスが順次に起動されるたびに、インデックス 01 から始まるログ ファイルが再作成されます。 このオプションを無効にすると、アプリケーションのインスタンスが（順次か同時かにかかわらず）起動されるたびに、最後に書き込まれたフォルダの後に続く新しいフォルダにトレース ファイルが保存されます。Cisco Unified JTAPI がトレース パスに存在する最後のフォルダを検出して、インデックスの値を自動的に増やします。
パス Trace Path（TracePath）	.	NA	NA	トレース ファイルを書き込むパスを指定します。このパスを指定しない場合は、デフォルトでアプリケーションのパスが使用されます。
ディレクトリ名ベース Directory Name Base（Directory）	.	NA	NA	トレース ファイルを格納するフォルダの名前を指定します。
ファイル名ベース File Name Base（FileNameBase）	Cisco Jtapi	NA	NA	この値はトレース ファイルの名前を作成するときに使用します。
ファイル名の拡張子 File Name Extension（FileNameExtension）	log	NA	NA	トレース ファイルの作成順を示す、ファイル ベース名の末尾に付加される数値インデックスを指定します。 [ファイル名ベース] フィールドに「jtapiTrace」と入力し、[ファイル名の拡張子] フィールドに「log」と入力した場合、トレース ファイルの名前は jtapiTrace01.log、jtapiTrace02.log のようになります。[ファイル名ベース] フィールドと [ファイル名の拡張子] フィールドを空のままにした場合、トレース ファイルの名前は CiscoJtapi01.log、CiscoJtapi02.log のようになります。

[CallManagers] タブ

このタブでは、オプションの Cisco Unified Communications Manager と接続するために JTAPI アプリケーションがユーザに提示する Cisco Unified Communications Manager のリストを定義できます。図 4-3 に、初期設定アプリケーションの [CallManagers] タブを示します。

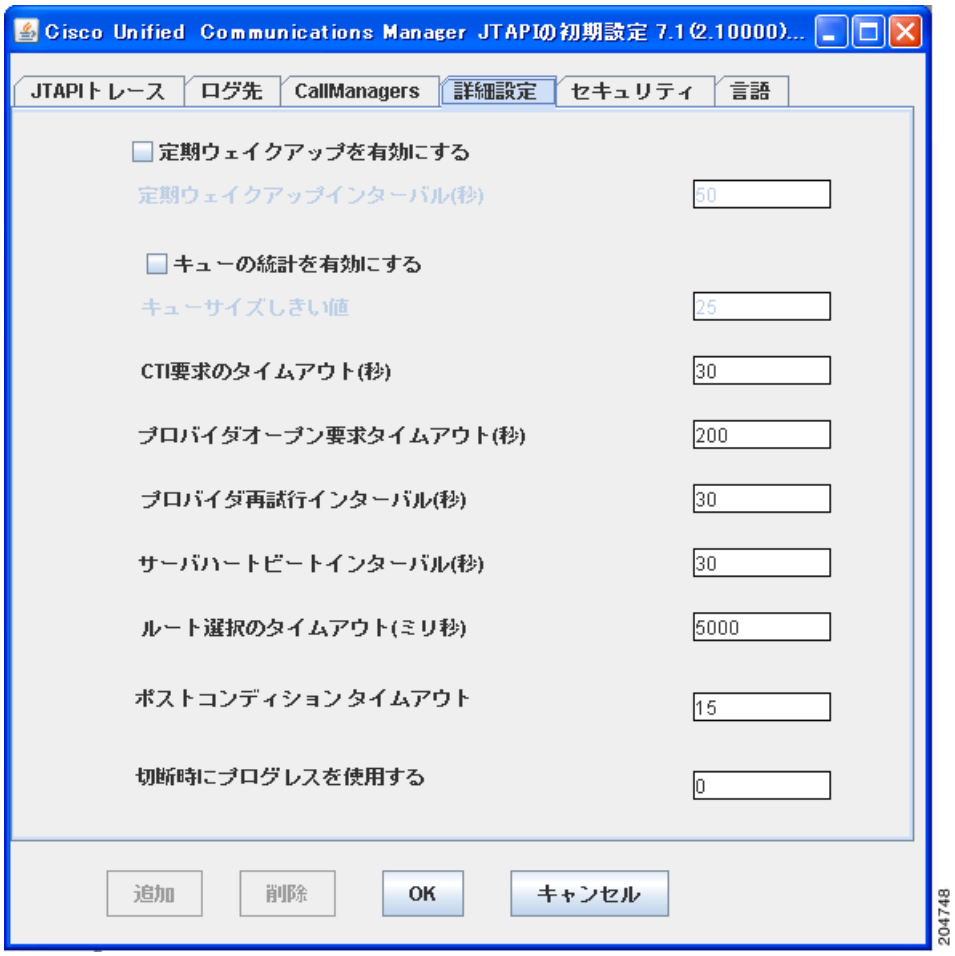
図 4-3　[CallManagers] タブ



[詳細設定] タブ

JTPrefs アプリケーションの [詳細設定] タブでは、表 4-4 のパラメータを設定できます。これらの低レベル パラメータの設定が必要になるのは、トラブルシューティングやデバッグを行う場合だけです。図 4-4 に、初期設定アプリケーションの [詳細設定] タブを示します。

図 4-4　　[詳細設定] タブ



（注） 表 4-4 のパラメータは、Cisco Technical Assistance Center（TAC）から指示された場合を除き、変更*しない*ことを強くお勧めします。

表 4-4　[詳細設定] タブの設定フィールド

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
定期ウェイクアップを有効にする Enable Periodic Wakeup （PeriodicWakeupEnabled）	FALSE （無効）	NA	NA	JTAPI が使用する内部メッセージ キューのハートビートを有効（または無効）にします。定期ウェイクアップインターバルで定義された時間内に JTAPI がメッセージを受信しなかった場合は、スレッドがアクティブになり、ログ イベントが作成されます。
定期ウェイクアップインターバル（秒） Periodic Wakeup Interval （PeriodicWakeupInterval）	50	NP	NP	JTAPI 内部メッセージ スレッドを非アクティブにする時間を秒数で定義します。この時間内に JTAPI がメッセージを受信しなかった場合は、スレッドがアクティブになり、イベントがログに記録されます。
キューの統計を有効にする Enable Queue Stats （QueueStatsEnabled）	FALSE （無効）	NA	NA	指定した数のメッセージが JTAPI メイン イベント スレッドにキューイングされるたびに最大のキュー項目数がログに記録されます。 x 個のメッセージが処理されるたびに、その期間における最大のキュー項目数を報告する DEBUGGING レベル トレースがログに記録されます（x は Queue Size Threshold で指定したメッセージの数を表します）。
キューサイズしきい値 Queue Size Threshold （QueueSizeThreshold）	25	10	NP	何個のメッセージを処理するたびに最大のキュー項目数を報告するかを指定します。
CTI 要求のタイムアウト（秒） CTI Request Timeout （CtiRequestTimeout）	15	10	NP	CTI 要求からの応答を待つ秒数を指定します。
プロバイダオープン要求タイムアウト（秒） Provider Open Request Timeout （ProviderOpenRequestTimeout）	200	10	NP	プロバイダー オープン要求に対する応答を待つ秒数を指定します。
プロバイダ再試行インターバル（秒） Provider Retry Interval （ProviderRetryInterval）	30	5	NP	システム障害が発生したときに Cisco Unified Communications Manager クラスタへの接続を再試行する秒数を指定します。
サーバハートビートインターバル（秒） Server Heartbeat Interval （DesiredServerHeartbeatInterval）	30	>0	NP	JTAPI と Cisco Unified Communications Manager クラスタが接続されていることを確認する間隔を秒単位で指定します。 この時間内にハートビートを受信しなかった場合、JTAPI はプロバイダー オープン要求で指定されている 2 番目の CTIManager を通じて接続を確立します。
ルート三択タイムアウト（ミリ秒） Route Select Timeout （RouteSelectTimeout）	5000	0	NP	ルート イベントに対するアプリケーションの応答を待つ時間をミリ秒単位で指定します。この時間内にアプリケーションが応答しなかった場合、JTAPI はそのルートを終了して、対応する RouteEnd イベントを送信します。

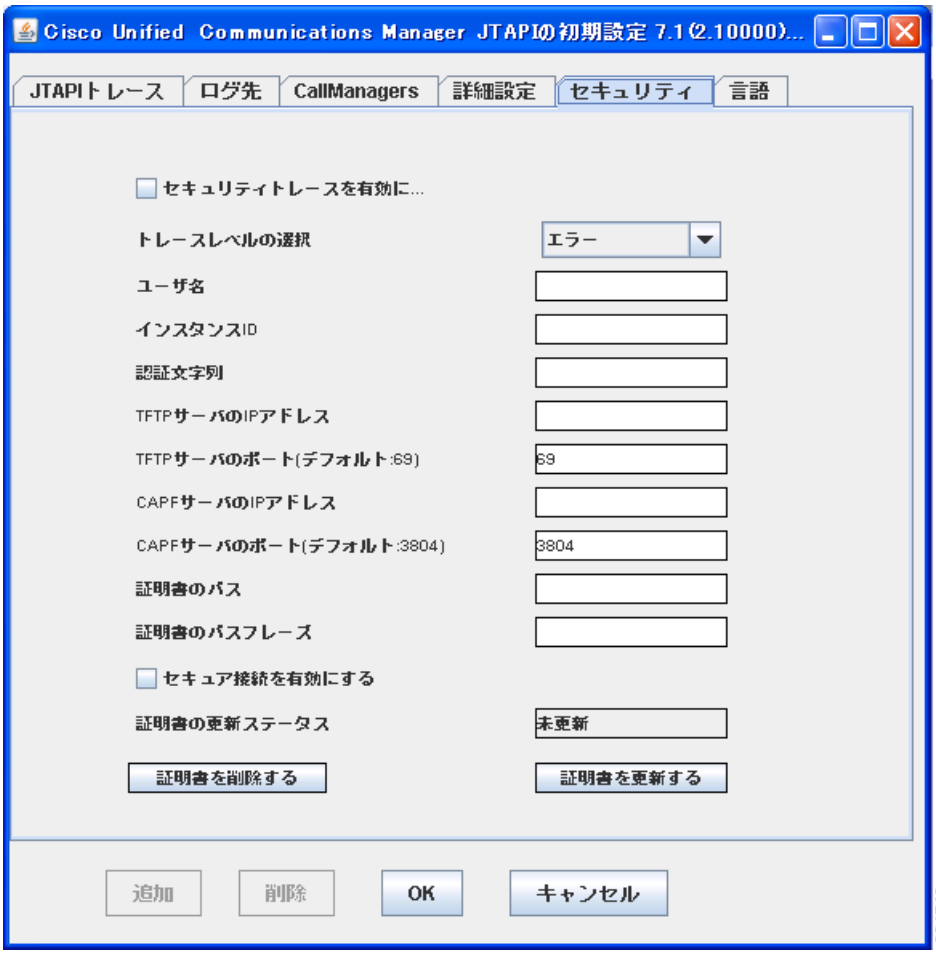
表 4-4　[詳細設定] タブの設定フィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
ポストコンディションタイムアウト Post Condition Timeout	15	0	NP	タイムアウトを指定します。
UseProgressAsDisconnectedDuringErrorEnabled	0	NA	NA	ゲートウェイから Progress を受け取った際に、Disconnect とみなす場合のコード。カンマ区切りのリストで記載します。たとえば、17（USERBUSY）と 25（EXCHANGEROUTINGERROR）の場合は、17, 15 と記載します。

[セキュリティ] タブ

図 4-5 に、初期設定アプリケーションの [セキュリティ] タブを示します。

図 4-5　[セキュリティ] タブ



アプリケーション ユーザは、JTAPI API または JTAPI Preferences を起動してアプリケーション サーバから証明書をダウンロードしてインストールする前に、JTAPI Preferences アプリケーションでユーザ名、インスタンス ID、許可コード、TFTP サーバの IP アドレス、CAPF サーバの IP アドレスの各パラメータを設定する必要があります。

JTAPI Preferences では、ユーザ名とインスタンス ID の 1 つ以上のペアに対してセキュリティ プロファイルを設定できます。ユーザ名とインスタンス ID のペアに対してすでにセキュリティ プロファイルが設定されている場合、アプリケーション ユーザがユーザ名とインスタンス ID を入力して他の編集ボックスをクリックすると、セキュリティ プロファイルが自動的に編集ボックスに入力されます。

JTAPI Preferences の GUI を使用せずに、CiscoJtapiProperties で提供されているインターフェイスをアプリケーションから呼び出してクライアント証明書をインストールすることもできます。UpdateCertificate() インターフェイスが呼び出されると、JTAPI クライアントは TFTP サーバに接続して CTL ファイルをダウンロードし、指定された証明書パスに証明書を抽出します。その後、CAPF サーバに接続してクライアント証明書をダウンロードし、指定された証明書パスにクライアント証明書をインストールします。

ユーザ セキュリティ レコードは Comma Separated Value（CSV; カンマ区切り形式）で jtapi.ini ファイルに保存されます。各レコードはセミコロンで区切られます。ユーザ セキュリティ レコードの例を次に示します。

SecurityProperty=user,123,12345,172.19.242.37,3804,172.19.242.37,69,.¥¥,true,false;<*次のレコード*>; …

[セキュリティ] タブでは、次のパラメータを設定できます。

表 4-5　JTAPI のセキュリティ設定のフィールド

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
セキュリティトレースを有効にする Enable Security Tracing（SecurityTraceEnabled）	FALSE	NA	NA	このチェックボックスをオンにしてトレース レベルを選択すると、証明書のインストール処理のトレースが有効（または無効）になります。
トレース レベルの選択 Select Trace Level（SecurityTraceLevel）	0	0	2	次の 3 つからトレース レベルを選択できます。 • Error = 0：エラー イベントをログに記録します。 • Debug = 1：デバッグ イベントをログに記録します。 • Detailed = 2：すべてのイベントをログに記録します。
ユーザ名 User Name（Username）	NA	NA	NA	ユーザ名とインスタンス ID のペアに対してすでにセキュリティ プロファイルが設定されている場合、アプリケーション ユーザがユーザ名とインスタンス ID を入力して他の編集ボックスをクリックすると、セキュリティ プロファイルが自動的に編集ボックスに入力されます。
インスタンス ID Instance ID（instanceID）	NA	NA	NA	このフィールドでアプリケーション インスタンスの ID を指定します。同じユーザ名を使用して CTIManager に接続するアプリケーションの場合は、インスタンスごとにインスタンス ID を定義して証明書の AuthorizationString をダウンロードする必要があります。

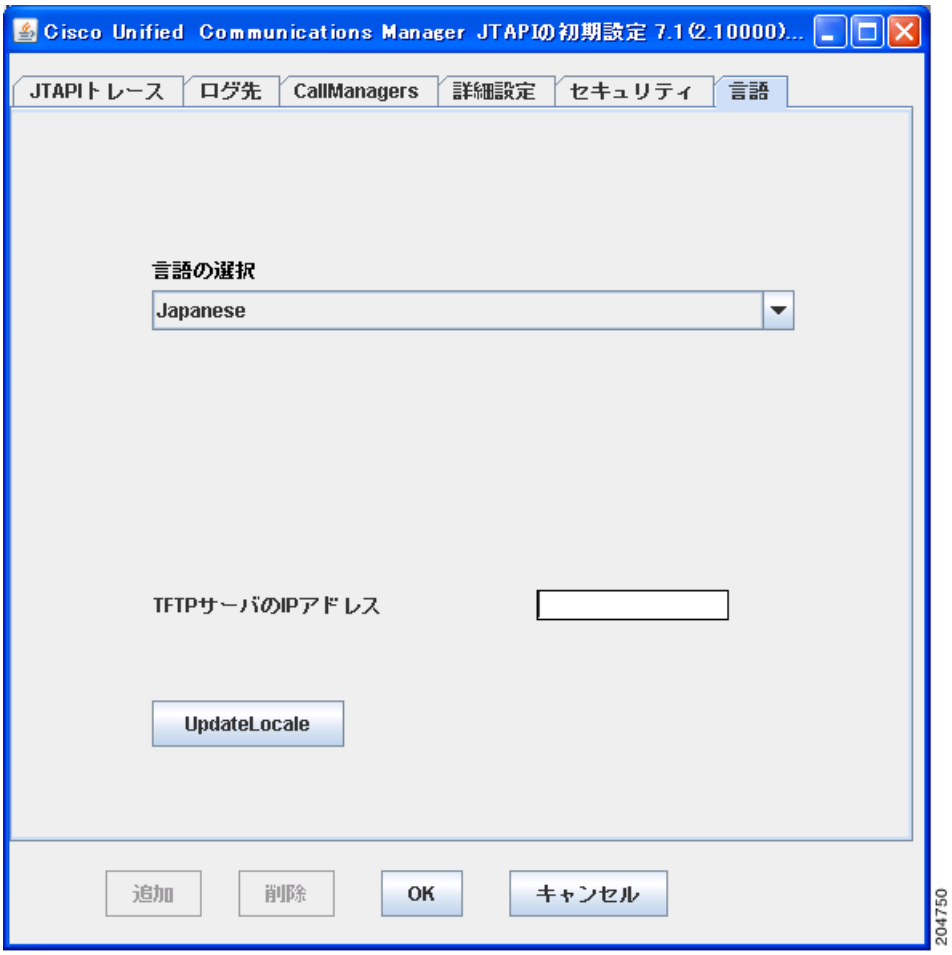
表 4-5　JTAPI のセキュリティ設定のフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
認証文字列 Authentication String（authcode）	NA	NA	NA	このフィールドで証明書のダウンロードに使用される 1 回限りの文字列を指定します。
TFTP サーバの IP アドレス TFTP Server IP Address	NA	NA	NA	このフィールドで TFTP サーバの IP アドレス（通常は Cisco Unified Communications Manager の IP アドレス）を指定します。
TFTP サーバのポート TFTP Server Port	69	NP	NP	TFTP サーバのポートはデフォルトで 69 です。システム管理者からの指示が限り、この値は変更しないでください。
CAPF サーバの IP アドレス CAPF Server IP Address	NA	NA	NA	このフィールドで CAPF サーバの IP アドレスをドット付き 10 進数で指定します。
CAPF サーバのポート CAPF Server Port	3804	NP	NP	CAPF サーバのポート番号はデフォルトで 3804 に設定されますが、この番号は Cisco Unified Communications Manager Administration で設定することもできます。JTAPI Preferences からこの値を入力する場合は、Cisco Unified Communications Manager Administration で設定した値と同じにする必要があります。
証明書のパス Certificate Path	JTAPI.jar location	NA	NA	このフィールドでアプリケーションがサーバ証明書とクライアント証明書をインストールするパスを指定します。このフィールドが空の場合、証明書は JTAPI.jar のクラスパスにインストールされます。
証明書のパスフレーズ Certificate Passphrase				このフィールドで証明書のパスフレーズを指定します。
セキュア接続を有効にする Enable Secure Connection	FALSE	NA	NA	このオプションをオンにすると、Cisco Unified Communications Manager へのセキュア TLS 接続が有効になります。このオプションをオフにした場合は、証明書が更新またはインストールされていても、JTAPI と CTI の接続は暗号化されません。
証明書更新ステータス Certificate Update Status	NA	NA	NA	このフィールドには、証明書の更新状態に関する情報が表示されます。
証明書を削除する Delete Certificate	NA	NA	NA	このボタンを使用すると、既存の証明書が削除されます。
証明書を更新する Update Certificate	NA	NA	NA	このボタンを使用すると、変更されたパラメータで既存の証明書が更新されます。

[言語] タブ

図 4-6 に、初期設定アプリケーションの [言語] タブを示します。

図 4-6　[言語] タブ



[言語] タブでは、システムにインストールされている言語の中から、設定の表示に使用する言語を 1 つ選択できます。選択可能な言語は次のとおりです。

Arabic（アラビア語）	Brazilian Portuguese（ブラジルポルトガル語）	Chinese Taiwan（繁体字中国語）	Croatian（クロアチア語）	Czech（チェコ語）
Danish（デンマーク語）	Dutch（オランダ語）	English（英語）	Finnish（フィンランド語）	French（フランス語）
German（ドイツ語）	Greek（ギリシャ語）	Hebrew（ヘブライ語）	Hungarian（ハンガリー語）	Italian（イタリア語）
Japanese（日本語）	Nederlands（オランダ語）	Norwegian（ノルウェー語）	Polish（ポーランド語）	Portuguese（ポルトガル語）
Russian（ロシア語）	Simplified Chinese（簡体字中国語）	Slovak（スロバキア語）	Spanish（スペイン語）	Swedish（スウェーデン語）

言語を選択すると、タブ内のテキストがその言語で表示されます。

JTAPI アプリケーションのユーザ情報の管理

JTAPI アプリケーションのユーザには、1 つ以上のデバイスを制御する特権が与えられている必要があります。JTAPI アプリケーションを使用する前に、「新規ユーザの追加」セクションの手順に従ってユーザを追加し、ユーザにデバイスを割り当ててください。ユーザに割り当てられているデバイスのリストには、ユーザがアプリケーションから制御（発呼や応答など）する必要がある電話機が表示されます。

jtapi.ini ファイルのフィールド

JTPrefs を起動できない非 GUI ベースのプラットフォームで実行されるアプリケーションの場合は、ここに示す値に基づく固有の jtapi.ini ファイルを作成して jtapi.jar とともに配置できます。これらの値は JTAPI の設定に使用されます。

各アプリケーションは 表 4-6 で説明されている有効なデータを提供する必要があります。jtapi.ini ファイルの値が不適切であるために JTAPI の動作で発生するエラーは、アプリケーション側に原因があります。

NA：適用なし
NP：なし

表 4-6　jtapi.ini ファイルのフィールド

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
INFORMATIONAL	0	NA	NA	ステータス イベントを指定します。
DEBUG	0	NA	NA	最高レベルのデバッグ イベントを指定します。
WARNING	0	NA	NA	低レベルの警告イベントを指定します。
JTAPI_DEBUGGING	0	NA	NA	JTAPI のメソッドおよびイベントのトレースを指定します。
JTAPIIMPL_DEBUGGING	0	NA	NA	内部 JTAPI 実装トレースを指定します。
CTI_DEBUGGING	0	NA	NA	JTAPI 実装に送信される Cisco Unified Communications Manager イベントのトレースを指定します。
CTIIMPL_DEBUGGING	0	NA	NA	内部 CTICLIENT 実装トレースを指定します。
PROTOCOL_DEBUGGING	0	NA	NA	CTI プロトコルの完全なデコードを指定します。
MISC_DEBUGGING	0	NA	NA	各種の低レベル デバッグ トレースを指定します。
DesiredServerHeartbeatInterval	30	>0	NP	このフィールドで JTAPI と Cisco Unified Communications Manager クラスタが接続されていることを確認する間隔を秒単位で指定します。この時間内にハートビートを受信しなかった場合、JTAPI はプロバイダー オープン要求で指定されている 2 番目の CTIManager を通じて接続を確立します。
TracePath	.	NA	NA	トレース ファイルを書き込むパス名を指定します。このパスを指定しない場合は、デフォルトでアプリケーションのパスが使用されます。

表 4-6　jtapi.ini ファイルのフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
FileNameExtension	log	NA	NA	このフィールドで、トレース ファイルの作成順を示すファイル ベース名の末尾に付加される数値インデックスを指定します。たとえば、[File Name Base] フィールドに「jtapiTrace」と入力し、[File Name Extension] フィールドに「log」と入力した場合、トレース ファイルの名前は jtapiTrace01.log、jtapiTrace02.log のようになり、jtapiTrace10.log まで達すると再び jtapiTrace01.log に書き込まれます。[File Name Base] フィールドと [File Name Extension] フィールドを空のままにした場合、トレース ファイルの名前は CiscoJtapi01.log、CiscoJtapi02.log のようになります。
SyslogCollector	FALSE	NA	NA	このフィールドで、システム上の特定のパスおよびフォルダにトレースを送信するように指定します。作成できるログ ファイルの数は 2 ～ 99 個です。ログ ファイルは番号順に書き込まれ、最後のファイルがいっぱいになると最初のファイルに戻ります。ログ ファイルのサイズは 1 MB ずつ増加します。
TraceFileSize	1048576	1048576	NP	このフィールドで、書き込むログ ファイルの最大サイズを指定します。
UseAlarmService	0	NA	NA	このオプションを有効にすると、指定したマシンで実行されているアラーム サービスに JTAPI アラームが送信されます。このオプションを有効にする場合は、ホスト名とポート番号を指定する必要があります。
ProviderOpenRequestTimeout	200	10	NP	このフィールドで、プロバイダー オープン要求に対する応答を待つ秒数を指定します。デフォルトは 10 秒です。
JtapiPostConditionTimeout	15	10	20	JTAPI にはイベントの事後条件があり、タイムアウト時間内に事後条件が満たされない場合は例外がスローされます。このフィールドでは、そのような条件のタイムアウト値を設定します。
ApplicationPriority	2	NA	NA	このフィールドは、複数のプロバイダー オープン要求の優先順位を設定するために使用します。現在のところ、JTAPI はデフォルト値しか送信しません。
SecurityTraceEnabled	FALSE	NA	NA	このフィールドで、セキュリティ関連メッセージのトレースを有効にします。 このチェックボックスを選択してトレース レベルを選択すると、証明書のインストール処理のトレースが有効（または無効）になります。
AlarmServicePort	1444	NP	NP	このフィールドは、アラームを別のサーバに送信するために使用します。アラーム サーバのホスト名とサービスが実行されているポートを選択すると、指定したサーバおよびポートにアラームが送信されます。
AlarmServiceHostname	null	NA	NA	このフィールドで、アラーム サーバのホスト名を表示します。

表 4-6　jtapi.ini ファイルのフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
RouteSelectTimeout	5000	0	NP	このフィールドで、Route イベントに対するアプリケーションの応答を待つ時間をミリ秒単位で指定します。この時間内にアプリケーションが応答しなかった場合、JTAPI はそのルートを終了して、対応する RouteEnd イベントを送信します。
ProviderRetryInterval	30	5	NP	このフィールドで、システム障害が発生したときに Cisco Unified Communications Manager クラスタへの接続を再試行する秒数を指定します。
QueueStatsEnabled	FALSE	NA	NA	このフィールドで、指定した数のメッセージが JTAPI メイン イベント スレッドにキューイングされるたびに JTAPI が最大のキュー項目数をログに記録するために使用します。つまり、x 個のメッセージが処理されるたびに、その期間における最大のキュー項目数を報告する DEBUGGING レベル トレースがログに記録されます（x は Queue Size Threshold で指定したメッセージの数を表します）。
FileNameBase	CiscoJtapi	NA	NA	このフィールドで、トレース ファイルの名前を作成するときの値を指定します。
PeriodicWakeupEnabled	FALSE	NA	NA	このフィールドで、JTAPI が使用する内部メッセージ キューのハートビートを有効（または無効）にします。PeriodicWakeupInterval で定義された時間内に JTAPI がメッセージを受信しなかった場合は、スレッドがアクティブになり、ログ イベントが作成されます。
JTAPINotificationPort	2789	1	NP	このフィールドで、実行中に JTAPI パラメータの変更を JTAPI アプリケーションに伝達するために使用するポートを指定します。
PeriodicWakeupInterval	50	NP	NP	このフィールドで、JTAPI 内部メッセージ スレッドを非アクティブにする時間を定義します。この時間内に JTAPI がメッセージを受信しなかった場合は、スレッドがアクティブになり、イベントがログに記録されます。
QueueSizeThreshold	25	10	NP	このフィールドで、何個のメッセージを処理するたびに最大のキュー項目数を報告するかを指定します。
UseSystemDotOut	FALSE	NA	NA	このフィールドは、コンソールにトレースを表示するために使用します。

表 4-6　jtapi.ini ファイルのフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
UseSameDirectory	1	NA	NA	このフィールドで、アプリケーションの各インスタンスで同じフォルダ名を使用する必要があるかどうかを指定します。 このオプションを有効にすると、同じディレクトリにログ ファイルがトレースされます。この場合は、JTAPI アプリケーションのインスタンスが順次に起動されるたびに、インデックス 01 から始まるログ ファイルが再作成されます。 このオプションを無効にすると、アプリケーションのインスタンスが（順次か同時かにかかわらず）起動されるたびに、最後に書き込まれたフォルダの後に続く新しいフォルダにトレース ファイルが保存されます。Cisco Unified JTAPI がトレース パスに存在する最後のフォルダを検出して、インデックスの値を自動的に増やします。
NumTraceFiles	10	2	1000	このフィールドで、書き込むログ ファイルの最大数を指定します。
UseSyslog	FALSE	NA	NA	このフィールドを有効にすると、Collector フィールドと Port Number フィールドで指定された UDP ポートにトレースが送信されます。トレースは syslog コレクタ サービスによって収集され、CiscoWorks2000 サーバに送信されます。
SecurityTraceLevel	0	0	2	このフィールドで、セキュリティ メッセージのトレース レベルを指定します。 0 = Error、1 = Debug、2 = Detailed
UseTraceFile	TRUE	NA	NA	このフィールドで、logFile Trace Writer へのログの書き込みを有効にします。
CMAssignedAppID	0	NA	NA	このフィールドで、アプリケーションに割り当てられている機能 ID を指定します。この ID は Cisco Unified Communications Manager によってあらかじめ割り当てられます。
UseProgressAsDisconnectedDuringErrorEnabled	0	NA	NA	ゲートウェイから Progress を受け取った際に、Disconnect とみなす場合のコード。カンマ区切りのリストで記載します。たとえば、17（USERBUSY）と 25（EXCHANGEROUTINGERROR）の場合は、17, 15 と記載します。
CtiManagers	null	NA	NA	このフィールドで、トレースを収集する CTI Manager のリストを指定します。
Directory	.	NA	NA	このフィールドで、トレース ファイルを格納するフォルダの名前を指定します。

表 4-6　jtapi.ini ファイルのフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
Security Property SecurityProperty=username, instanceId, authcode, tftp ip address, tftp port, capf ip address, capf port, certificate path, security option, certificate status	NA	NA	NA	このフィールドで、ユーザ セキュリティ レコード（username、instanceId、authcode、tftp ip address、tftp port、capf ip address、capf port、certificate path、security option、certificate status）を指定します。このレコードは、jtapi.ini ファイルにカンマ区切りの文字列で保存されます。各レコードはセミコロンで区切られます。 SecurityProperty=user,123,12345,172.19.242.37,3804,172.19.242.37,69,.¥¥,true,false;<次のレコード>;…
Username	NA	NA	NA	このフィールドには、以前にユーザ名とインスタンス ID のペアが設定され、他の編集ボックスをクリックしたアプリケーション ユーザーのセキュリティ プロファイルが自動的に入力されます。
instanceId	NA	NA	NA	このフィールドでアプリケーション インスタンスの ID を指定します。同じユーザ名を使用して CTIManager に接続するアプリケーションの場合は、インスタンスごとにインスタンス ID を定義して証明書の AuthorizationString をダウンロードする必要があります。
authcode	NA	NA	NA	このフィールドで、Cisco Unified Communications Manager データベースで設定された認証文字列を指定します。これは証明書を取得するために一度だけ使用できます。
Communications Manager TFTP IP address	NA	NA	NA	このフィールドで、Cisco Unified Communications Manager の TFTP アドレス（通常は Cisco Unified Communications Manager の IP アドレス）を指定します。
CallManger TFTP port	69	NP	NP	このフィールドには CallManager TFTP port が表示されます。 システム管理者からの指示がない限り、この値はデフォルトの 69 から変更しないでください。
Communications Manager CAPF IP server address	NA	NA	NA	このフィールドで、CAPF サーバの IP アドレスを指定します。
Communications Manager CAPF server port	3804	NP	NP	このフィールドには、CAPF サーバ ポートのデフォルト値（3804）が表示されます。この値は Cisco Unified Communications Manager Administration のサービス パラメータで設定できることに注意してください。このインターフェイスから値を入力する場合は、Cisco Unified Communications Manager Administration ウィンドウで設定した値と同じであることを確認してください。
Certificate path	JTAPI.jar の場所	NA	NA	このフィールドで、アプリケーションがサーバ証明書とクライアント証明書をインストールする場所を指定します。このフィールドが空の場合、証明書は JTAPI.jar のクラスパスにインストールされます。

表 4-6　jtapi.ini ファイルのフィールド （続き）

jtapi.ini のフィールド	デフォルト	最小値	最大値	説明
Enable secure connection	TRUE	NA	NA	このフィールドを TRUE に設定した場合は、証明書が更新またはインストールされていても、JTAPI と CTI の接続は暗号化されません。
Certificate Update Status	NA	NA	NA	[JTAPI Preferences] ダイアログボックスは、ユーザ名とインスタンス ID の 1 つ以上のペアに対してセキュリティ プロファイルを設定するために使用します。

デフォルト値を使用する場合の jtapi.ini ファイルの例

```
#Cisco Unified JTAPI version 7.0(1.1000)-1 Release ini parameters
#Wed Sep 14 16:55:30 PDT 2008
INFORMATIONAL=0
DesiredServerHeartbeatInterval=30
TracePath=.
FileNameExtension=log
SyslogCollector=
TraceFileSize=1048576
UseAlarmService=0
ProviderOpenRequestTimeout=200
JtapiPostConditionTimeout=15
ApplicationPriority=2
SecurityTraceEnabled=0
AlarmServicePort=1444
RouteSelectTimeout=5000
ProviderRetryInterval=30
QueueStatsEnabled=0
FileNameBase=CiscoJtapi
JTAPI_DEBUGGING=0
PeriodicWakeupEnabled=0
CTI_DEBUGGING=0
JTAPINotificationPort=2789
Traces=WARNING;INFORMATIONAL;DEBUG
PeriodicWakeupInterval=50
AlarmServiceHostname=
QueueSizeThreshold=25
Debugging=JTAPI_DEBUGGING;JTAPIIMPL_DEBUGGING;CTI_DEBUGGING;CTIIMPL_DEBUGGING;
PROTOCOL_DEBUGGING;MISC_DEBUGGING
PROTOCOL_DEBUGGING=0
UseSystemDotOut=0
MISC_DEBUGGING=0
UseSameDirectory=1
NumTraceFiles=10
UseSyslog=0
DEBUG=0
SecurityTraceLevel=0
UseTraceFile=1
WARNING=0
CMAssignedAppID=0
UseProgressAsDisconnectedDuringErrorEnabled=0
CtiManagers=;;;;;;;;;
Directory=
CTIIMPL_DEBUGGING=0
CtiRequestTimeout=30
JTAPIIMPL_DEBUGGING=0
SyslogCollectorUDPPort=514
SecurityProperty=cisco,123,12345,A.B.C.D,3804,A.B.C.D,69,/C¥:/Program
Files/JTAPITools/./,false,false;
```

CHAPTER 5

Cisco Unified JTAPI パッケージのすべての階層

使用可能な Cisco Unified JTAPI パッケージは次のとおりです。

- com.cisco.jtapi.extensions：この拡張の実装情報については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。
- com.cisco.services.alarm：アラーム クラスの実装情報については、第 7 章「Cisco Unified JTAPI のアラームとサービス」を参照してください。
- com.cisco.services.tracing：トレースの実装情報については、第 7 章「Cisco Unified JTAPI のアラームとサービス」を参照してください。
- com.cisco.services.tracing.implementation：トレースの実装情報については、第 7 章「Cisco Unified JTAPI のアラームとサービス」を参照してください。

クラスの階層

java.lang.Object

- com.cisco.services.alarm.AlarmManager
- com.cisco.services.tracing.BaseTraceWriter（com.cisco.services.tracing.TraceWriter を実装）
  - com.cisco.services.tracing.ConsoleTraceWriter
  - com.cisco.services.tracing.LogFileTraceWriter
  - com.cisco.services.tracing.OutputStreamTraceWriter
  - com.cisco.services.tracing.SyslogTraceWriter
- com.cisco.jtapi.extensions.CiscoAddressCallInfo
- com.cisco.jtapi.extensions.CiscoJtapiVersion
- com.cisco.jtapi.extensions.CiscoMediaCapability
  - com.cisco.jtapi.extensions.CiscoG711MediaCapability
  - com.cisco.jtapi.extensions.CiscoG723MediaCapability
  - com.cisco.jtapi.extensions.CiscoG729MediaCapability
  - com.cisco.jtapi.extensions.CiscoGSMMediaCapability
  - com.cisco.jtapi.extensions.CiscoWideBandMediaCapability
- com.cisco.jtapi.extensions.CiscoRTPParams

com.cisco.services.alarm.DefaultAlarm（com.cisco.services.alarm.Alarm を実装）

com.cisco.services.alarm.DefaultAlarmWriter（com.cisco.services.alarm.AlarmWriter を実装）

com.cisco.services.alarm.ParameterList

java.lang.Throwable（java.io.Serializable を実装）

java.lang.Exception

com.cisco.jtapi.extensions.CiscoRegistrationException

com.cisco.jtapi.extensions.CiscoUnregistrationException

com.cisco.services.tracing.TraceManagerFactory

com.cisco.services.tracing.implementation.TraceManagerImpl（com.cisco.services.tracing.TraceManager を実装）

com.cisco.services.tracing.implementation.TraceWriterManagerImpl（com.cisco.services.tracing.TraceWriterManager を実装）

インターフェイスの階層

javax.telephony.Address

com.cisco.jtapi.extensions.CiscoAddress（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

com.cisco.jtapi.extensions.CiscoIntercomAddress

javax.telephony.callcenter.RouteAddress

com.cisco.jtapi.extensions.CiscoRouteAddress

javax.telephony.AddressObserver

com.cisco.jtapi.extensions.CiscoAddressObserver

com.cisco.services.alarm.Alarm

com.cisco.services.alarm.AlarmWriter

javax.telephony.Call

javax.telephony.callcontrol.CallControlCall

com.cisco.jtapi.extensions.CiscoCall（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

com.cisco.jtapi.extensions.CiscoConsultCall

com.cisco.jtapi.extensions.CiscoCallCtlTermConnHeldReversionEv

com.cisco.jtapi.extensions.CiscoConferenceChain

com.cisco.jtapi.extensions.CiscoFeatureReason

com.cisco.jtapi.extensions.CiscoJtapiException

com.cisco.jtapi.extensions.CiscoJtapiProperties

com.cisco.jtapi.extensions.CiscoLocales

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMediaConnectionMode

com.cisco.jtapi.extensions.CiscoMediaEncryptionAlgorithmType

com.cisco.jtapi.extensions.CiscoMediaEncryptionKeyInfo

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMonitorInitiatorInfo

com.cisco.jtapi.extensions.CiscoMonitorTargetInfo

com.cisco.jtapi.extensions.CiscoObjectContainer

- com.cisco.jtapi.extensions.CiscoAddress（javax.telephony.Address も拡張）
  - com.cisco.jtapi.extensions.CiscoIntercomAddress
- com.cisco.jtapi.extensions.CiscoCall（javax.telephony.callcontrol.CallControlCall も拡張）
- com.cisco.jtapi.extensions.CiscoConsultCall
- com.cisco.jtapi.extensions.CiscoCallID
- com.cisco.jtapi.extensions.CiscoConnection（javax.telephony.callcontrol.CallControlConnection も拡張）
- com.cisco.jtapi.extensions.CiscoConnectionID
- com.cisco.jtapi.extensions.CiscoConsultCall
- com.cisco.jtapi.extensions.CiscoIntercomAddress
- com.cisco.jtapi.extensions.CiscoJtapiPeer（javax.telephony.JtapiPeer、com.cisco.services.tracing.TraceModule も拡張）
- com.cisco.jtapi.extensions.CiscoMediaTerminal
- com.cisco.jtapi.extensions.CiscoProvider
- com.cisco.jtapi.extensions.CiscoRouteTerminal
- com.cisco.jtapi.extensions.CiscoTerminal（javax.telephony.Terminal も拡張）
  - com.cisco.jtapi.extensions.CiscoMediaTerminal
  - com.cisco.jtapi.extensions.CiscoRouteTerminal
- com.cisco.jtapi.extensions.CiscoTerminalConnection（javax.telephony.callcontrol.CallControlTerminalConnection も拡張）

com.cisco.jtapi.extensions.CiscoPartyInfo

com.cisco.jtapi.extensions.CiscoProvFeatureID

com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoRecorderInfo

com.cisco.jtapi.extensions.CiscoRTPBitRate

com.cisco.jtapi.extensions.CiscoRTPInputProperties

com.cisco.jtapi.extensions.CiscoRTPOutputProperties

com.cisco.jtapi.extensions.CiscoRTPPayload

com.cisco.jtapi.extensions.CiscoSynchronousObserver

com.cisco.jtapi.extensions.CiscoTermConnPrivacyChangedEv

com.cisco.jtapi.extensions.CiscoTermEvFilter

com.cisco.jtapi.extensions.CiscoTerminalProtocol com.cisco.jtapi.extensions.CiscoTone
com.cisco.jtapi.extensions.CiscoUrlInfo

javax.telephony.Connection

javax.telephony.callcontrol.CallControlConnection

com.cisco.jtapi.extensions.CiscoConnection
（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

javax.telephony.events.Ev

javax.telephony.events.AddrEv

com.cisco.jtapi.extensions.CiscoAddrEv（com.cisco.jtapi.extensions.CiscoEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrRecordingConfigChangedEv

javax.telephony.callcontrol.events.CallCtlEv

javax.telephony.callcontrol.events.CallCtlCallEv（javax.telephony.events.CallEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnEv（javax.telephony.events.ConnEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv

javax.telephony.events.CallEv

javax.telephony.events.CallActiveEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv
（com.cisco.jtapi.extensions.CiscoCallEv も拡張）

javax.telephony.callcontrol.events.CallCtlCallEv
（javax.telephony.callcontrol.events.CallCtlEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnEv（javax.telephony.events.ConnEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallEv（com.cisco.jtapi.extensions.CiscoEv も拡張）

com.cisco.jtapi.extensions.CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv
（javax.telephony.events.CallActiveEv も拡張）

com.cisco.jtapi.extensions.CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferStartEv

javax.telephony.events.ConnEv

javax.telephony.callcontrol.events.CallCtlConnEv
（javax.telephony.callcontrol.events.CallCtlCallEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv

javax.telephony.events.TermConnEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringEndEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringStartEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorInitiatorInfoEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingEndEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingStartEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnSelectChangedEv

com.cisco.jtapi.extensions.CiscoEv

com.cisco.jtapi.extensions.CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrCreatedEv

com.cisco.jtapi.extensions.CiscoAddrEv（javax.telephony.events.AddrEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoAddrEv、com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoAddrEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallEv（javax.telephony.events.CallEv も拡張）

com.cisco.jtapi.extensions.CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv（javax.telephony.events.CiscoCallEv も拡張）

com.cisco.jtapi.extensions.CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferStartEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv（javax.telephony.events.CallActiveEv、com.cisco.jtapi.extensions.CiscoCallEv も拡張）

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoOutOfServiceEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv（com.cisco.jtapi.extensions.CiscoAddrEv も拡張）

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv（com.cisco.jtapi.extensions.CiscoTermEv も拡張）

com.cisco.jtapi.extensions.CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvFeatureEv（javax.telephony.events.ProvEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv

com.cisco.jtapi.extensions.CiscoAddrCreatedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvFeatureEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoTermActivatedEv
com.cisco.jtapi.extensions.CiscoTermCreatedEv
com.cisco.jtapi.extensions.CiscoTermRemovedEv
com.cisco.jtapi.extensions.CiscoTermRestrictedEv
com.cisco.jtapi.extensions.CiscoProvFeatureEv
com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
com.cisco.jtapi.extensions.CiscoTermActivatedEv
com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
com.cisco.jtapi.extensions.CiscoTermCreatedEv
com.cisco.jtapi.extensions.CiscoTermDataEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv
com.cisco.jtapi.extensions.CiscoTermEvFilter（javax.telephony.events.TermEv も拡張）
com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv
com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
com.cisco.jtapi.extensions.CiscoTermDataEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張）

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoOutOfServiceEv、com.cisco.jtapi.extensions.CiscoTermEv も拡張）

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermRemovedEv

com.cisco.jtapi.extensions.CiscoTermRestrictedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferStartEv

javax.telephony.events.ProvEv

com.cisco.jtapi.extensions.CiscoProvEv（com.cisco.jtapi.extensions.CiscoEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrCreatedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvFeatureEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoTermActivatedEv

com.cisco.jtapi.extensions.CiscoTermCreatedEv

com.cisco.jtapi.extensions.CiscoTermRemovedEv

com.cisco.jtapi.extensions.CiscoTermRestrictedEv

javax.telephony.events.TermEv

com.cisco.jtapi.extensions.CiscoTermEv（com.cisco.jtapi.extensions.CiscoEv も拡張）

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv

com.cisco.jtapi.extensions.CiscoRTPInputStartedEv

com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv

com.cisco.jtapi.extensions.CiscoTermButtonPressedEv

com.cisco.jtapi.extensions.CiscoTermDataEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv
（com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張）

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

javax.telephony.JtapiPeer

com.cisco.jtapi.extensions.CiscoJtapiPeer（com.cisco.jtapi.extensions.CiscoObjectContainer、com.cisco.services.tracing.TraceModule も拡張）

javax.telephony.Provider

com.cisco.jtapi.extensions.CiscoProvider（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

javax.telephony.capabilities.ProviderCapabilities

com.cisco.jtapi.extensions.CiscoProviderCapabilities

javax.telephony.ProviderObserver

com.cisco.jtapi.extensions.CiscoProviderObserver

javax.telephony.callcenter.RouteSession

com.cisco.jtapi.extensions.CiscoRouteSession

javax.telephony.callcenter.events.RouteSessionEvent

javax.telephony.callcenter.events.RouteEvent

com.cisco.jtapi.extensions.CiscoRouteEvent

javax.telephony.callcenter.events.RouteUsedEvent

com.cisco.jtapi.extensions.CiscoRouteUsedEvent

javax.telephony.Terminal

com.cisco.jtapi.extensions.CiscoTerminal（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

com.cisco.jtapi.extensions.CiscoMediaTerminal

com.cisco.jtapi.extensions.CiscoRouteTerminal

javax.telephony.TerminalConnection

javax.telephony.callcontrol.CallControlTerminalConnection

com.cisco.jtapi.extensions.CiscoTerminalConnection（com.cisco.jtapi.extensions.CiscoObjectContainer も拡張）

javax.telephony.TerminalObserver

com.cisco.jtapi.extensions.CiscoTerminalObserver

com.cisco.services.tracing.Trace

com.cisco.services.tracing.ConditionalTrace

com.cisco.services.tracing.UnconditionalTrace

com.cisco.services.tracing.TraceManagercom.cisco.services.tracing.TraceModule

com.cisco.jtapi.extensions.CiscoJtapiPeer（com.cisco.jtapi.extensions.CiscoObjectContainer、javax.telephony.JtapiPeer も拡張）

com.cisco.services.tracing.TraceWriter

com.cisco.services.tracing.TraceWriterManager

CHAPTER 6

Cisco Unified JTAPI 拡張

Cisco Unified JTAPI 拡張は、JTAPI 1.2 仕様では提供されていないものの、Cisco Unified Communications Manager の機能を利用するためのクラスとインターフェイスのセットで構成されます。開発者はこの拡張を使用して、新しいアプリケーションを作成することも、既存の拡張を変更して新しいメソッドを作成することもできます。

この章では、Cisco Unified Communications Manager で利用できる拡張 (インターフェイスとクラス) について説明し、次のセクションで構成されています。

クラスの階層

次のクラス階層は com.cisco.jtapi.extensions パッケージに含まれています。

hierarchy.java.lang.Object

- com.cisco.jtapi.extensions.CiscoAddressCallInfo
- com.cisco.jtapi.extensions.CiscoJtapiVersion
- com.cisco.jtapi.extensions.CiscoMediaCapability
  - com.cisco.jtapi.extensions.CiscoG711MediaCapability
  - com.cisco.jtapi.extensions.CiscoG723MediaCapability
  - com.cisco.jtapi.extensions.CiscoG729MediaCapability
  - com.cisco.jtapi.extensions.CiscoGSMMediaCapability
  - com.cisco.jtapi.extensions.CiscoWideBandMediaCapability
- com.cisco.jtapi.extensions.CiscoRTPParams
- java.lang.Throwable（java.io.Serializable を実装）
  - java.lang.Exception
    - com.cisco.jtapi.extensions.CiscoRegistrationException
    - com.cisco.jtapi.extensions.CiscoUnregistrationException

CiscoAddressCallInfo

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1 (2)	変更を記録するために履歴表に追加されました。

宣言

public class CiscoAddressCallInfo extends java.lang.Object

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoAddressCallInfo

コンストラクタ

CiscoAddressCallInfo (int inumActiveCalls, int imaxActiveCalls, int inumCallsOnHold, int imaxCallsOnHold)

CiscoAddressCallInfo (int inumActiveCalls, int imaxActiveCalls, int inumCallsOnHold, int imaxCallsOnHold, CiscoCall[] icalls)

フィールド

なし

メソッド

表 6-1　CiscoAddressCallInfo のメソッド

インターフェイス	メソッド	説明
CiscoCall[]	getCalls()	CiscoAddress に対する Cisco コールの配列を返します。
int	getMaxActiveCalls()	CiscoAddress でサポートされるアクティブ コールの最大数を整数で返します。
int	getMaxCallsOnHold()	CiscoAddress で保留状態にできるコールの最大数を整数で返します。
int	getNumActiveCalls()	CiscoAddress のアクティブ コールの数を整数で返します。
int	getNumCallsOnHold()	CiscoAddress で保留中のコールの数を整数で返します。

継承したメソッド

クラス java.lang.Object から
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

関連資料

なし

CiscoG711MediaCapability

CiscoG711MediaCapability オブジェクトは G.711 で符号化された RTP ストリームのプロパティを指定します。G.711 メディア終端をサポートしているアプリケーションは、CiscoMediaTerminal を登録する際に、このオブジェクトを使用して適切なパケット サイズを指定します。デフォルトのパケット サイズは 30 ミリ秒です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1(x)	変更を記録するために履歴表に追加されました。

宣言

public class CiscoG711MediaCapability extends CiscoMediaCapability

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoMediaCapability

　　　　com.cisco.jtapi.extensions.CiscoG711MediaCapability

コンストラクタ

表 6-2　CiscoG711MediaCapability のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoG711MediaCapability(int rtpPacketFrameSize)	CiscoG711MediaCapability を構築します。
public	CiscoG711MediaCapability()	CiscoG711MediaCapability を構築します。

フィールド

表 6-3　CiscoG711MediaCapability のフィールド

インターフェイス	フィールド	説明
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：20 ミリ秒の RTP パケット。
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：30 ミリ秒の RTP パケット。
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：60 ミリ秒の RTP パケット。

継承したフィールド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

メソッド

なし

継承したメソッド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
getMaxFramesPerPacket, getPayloadType, isSupported, toString

クラス java.lang.Object から
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoG723MediaCapability

CiscoG723MediaCapability オブジェクトは G.723 で符号化された RTP ストリームのプロパティを指定します。G.723 メディア終端をサポートしているアプリケーションは、CiscoMediaTerminal を登録する際に、このオブジェクトを使用して適切なパケット サイズおよびビット レートを指定します。デフォルトのパケット サイズは 30 ミリ秒で、デフォルトのビット レートは 6.4k です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoG723MediaCapability extends CiscoMediaCapability

java.lang.Object

com.cisco.jtapi.extensions.CiscoMediaCapability

com.cisco.jtapi.extensions.CiscoG723MediaCapability

コンストラクタ

表 6-4　CiscoG723MediaCapability のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoG723MediaCapability(int rtpPacketFrameSize , int bitRate)	CiscoG723MediaCapability を構築します

フィールド

表 6-5　CiscoG723MediaCapability のフィールド

インターフェイス	フィールド	説明
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：20 ミリ秒の RTP パケット。
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：30 ミリ秒の RTP パケット。
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：60 ミリ秒の RTP パケット。

継承したフィールド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

メソッド

表 6-6　CiscoG723MediaCapability のメソッド

インターフェイス	メソッド	説明
public int	getBitRate()	この機能オブジェクトで指定されたビット レートを返します。戻り値：RTPBitRate インターフェイスからのビット レート。
public java.lang.String	toString()	Object.toString() メソッドをオーバーライドします。オーバーライド：クラス CiscoMediaCapability の toString。

継承したメソッド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
getMaxFramesPerPacket, getPayloadType, isSupported

クラス java.lang.Object から
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoG729MediaCapability

CiscoG729MediaCapability オブジェクトは G.729 で符号化された RTP ストリームのプロパティを指定します。G.729 メディア終端をサポートしているアプリケーションは、CiscoMediaTerminal を登録する際に、このオブジェクトを使用して適切なパケット サイズを指定します。デフォルトのパケット サイズは 30 ミリ秒です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoG729MediaCapability extends CiscoMediaCapability

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoMediaCapability

　　　　com.cisco.jtapi.extensions.CiscoG729MediaCapability

コンストラクタ

表 6-7　G729MediaCapability のコンストラクタ

コンストラクタ	説明
CiscoG729MediaCapability(int payload, int rtpPacketFrameSize)	CiscoG729MediaCapability を構築します。

フィールド

表 6-8　CiscoG729MediaCapability のフィールド

インターフェイス	フィールド	説明
static int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：60 ミリ秒の RTP パケット。
static int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：30 ミリ秒の RTP パケット。
static int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：20 ミリ秒の RTP パケット。
static int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：20 ミリ秒の RTP パケット。

継承したフィールド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

メソッド

なし

継承したメソッド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から

getMaxFramesPerPacket, getPayloadType, isSupported, toString

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoGSMMediaCapability

CiscoGSMMediaCapability オブジェクトは GSM で符号化された RTP ストリームのプロパティを指定します。GSM メディア終端をサポートしているアプリケーションは、CiscoMediaTerminal を登録する際に、このオブジェクトを使用して適切なパケット サイズを指定します。デフォルトのパケット サイズは 30 ミリ秒です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoGSMMediaCapability extends CiscoMediaCapability

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoMediaCapability

　　　　com.cisco.jtapi.extensions.CiscoGSMMediaCapability

コンストラクタ

表 6-9　CiscoGSMMediaCapability のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoGSMMediaCapability()	CiscoGSMMediaCapability を構築します。
public	CiscoGSMMediaCapability(int rtpPacketFrameSize)	CiscoGSMMediaCapability を構築します。

フィールド

表 6-10　CiscoGSMMediaCapability のフィールド

インターフェイス	フィールド	説明
static int	FRAMESIZE_EIGHTY_MILLISECOND_PACKET	RTP パケットのフレームサイズ：80 ミリ秒の RTP パケット。

継承したフィールド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

メソッド

なし

継承したメソッド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から
getMaxFramesPerPacket, getPayloadType, isSupported, toString

クラス java.lang.Object から
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

なし

CiscoJtapiVersion

このクラスは、インストールされている Cisco JTAPI のバージョン情報を提供します。プログラムはアクセス用メソッドを使用してバージョン番号を取得できます。Cisco Jtapi Version は a.b(x.y) の形式で提供されます（a はメジャー バージョン、b はマイナー バージョン、x はリビジョン番号、y はビルド番号を表します）。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoJtapiVersion extends java.lang.Object

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoJtapiVersion

コンストラクタ

publicCiscoJtapiVersion()None

フィールド

なし

メソッド

表 6-11　CiscoJtapiVersion のメソッド

インターフェイス	メソッド	説明
java.lang.String	getBuildDescription()	リリース バージョンの場合は「release」を返し、リリース バージョンでない場合は「debug」を返します。
int	getBuildNumber()	バージョンのビルド番号を返します。
int	getExtendedBuildNumber()	バージョンの拡張ビルド番号を返します。
int	getMajorVersion()	メジャー バージョン番号を返します。
int	getMinorVersion()	マイナー バージョン番号を返します。
int	getRevisionNumber()	バージョンのリビジョン番号を返します。
public java.lang.String	getVersion()	名前を使用せず、a.b(x.y)-z の形式でバージョン情報を返します。
public java.lang.String	toString()	a.b(x.y)-z の形式でバージョン情報を返します。クラス java.lang.Object の toString をオーバーライドします。

継承したメソッド

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

なし

CiscoMediaCapability

CiscoMediaCapability オブジェクトは、登録された CiscoMediaTerminals に対してアプリケーションがサポートできるメディア形式のプロパティを指定します。CiscoMediaCapability は抽象クラスなので、アプリケーションはそのサブクラスだけを直接的に構築できます。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoMediaCapability extends java.lang.Object

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoMediaCapability

サブクラス

CiscoG711MediaCapability, CiscoG723MediaCapability, CiscoG729MediaCapability, CiscoGSMMediaCapability, CiscoWideBandMediaCapability

コンストラクタ

表 6-12　CiscoMediaCapability のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoMediaCapability(int payloadType, int maxFramesPerPacket)	ペイロード タイプとパケットサイズを指定して CiscoMediaCapability オブジェクトを構築します（ミリ秒単位）。

フィールド

表 6-13　CiscoMediaCapability のフィールド

インターフェイス	フィールド	説明
static	G711_64K_30_MILLISECONDS	デフォルト パラメータを使用する G.711 機能。
static	G723_6K_30_MILLISECONDS	デフォルト パラメータを使用する G.723 機能。
static	G729_30_MILLISECONDS	デフォルト パラメータを使用する G.729 機能。
static	GSM_80_MILLISECONDS	デフォルト パラメータを使用する GSM 機能。
static	WIDEBAND_256K_10_MILLISECONDS	デフォルト パラメータを使用するワイドバンド機能。

メソッド

表 6-14　CiscoMediaCapability のメソッド

インターフェイス	メソッド	説明
int	getMaxFramesPerPacket(	このオブジェクトで指定されているパケット サイズ（ミリ秒単位）を返します。maxFramesPerPacket パラメータは、H.245 プロトコル定義を受け継いだものです。 Cisco Unified Communications Manager ではこのフィールドを、RTP パケットあたりのフレーム数ではなく、デバイスが受信可能な RTP パケットあたりのオーディオ（ミリ秒単位）として使用します。 Cisco Unified IP Phone での発着信ではこのレートを超えることができませんが、サード パーティ製の IP フォンでは、他の（より高い）レートを使用できる場合があります。
int	getPayloadType()	このオブジェクトで指定されている RTPPayload インターフェイスのペイロード タイプを返します。
boolean	isSupported()	このオブジェクトのペイロードがサポートされているかどうかを返します。payloadType がサポートされている場合に true、そうでない場合に false を返します。
java.lang.String	toString()	クラス java.lang.Object の toString をオーバーライドします。

継承したメソッド

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

CiscoG711MediaCapability、CiscoG723MediaCapability、CiscoG729MediaCapability、CiscoGSMMediaCapability、CiscoWideBandMediaCapability、CiscoRTPBitRate、および CiscoRTPPayload を参照してください。

CiscoRegistrationException

CiscoMediaTerminal.register メソッドは、登録プロセスが失敗すると、その理由にかかわらず、この例外をスローします。たとえば、プロバイダーが OUT_OF_SERVICE である場合、またはデバイスが登録済みである場合、登録は失敗します。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoRegistrationException extends java.lang.Exception

java.lang.Object

　java.lang.Throwable

　　java.lang.Exception

　　com.cisco.jtapi.extensions.CiscoRegistrationException

実装インターフェイス

java.io.Serializable

コンストラクタ

表 6-15　CiscoRegistrationException のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoRegistrationException(java.lang.String description)	例外の説明をパラメータとして使用します。

メソッド

なし

継承したメソッド

クラス java.lang.Throwable から

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

CiscoMediaTerminal.register(java.net.InetAddress, int, com.cisco.jtapi.extensions.CiscoMediaCapability[]) を参照してください。

CiscoRTPParams

CiscoRTPParams クラスを使用して、コールごとにメディア端末の動的 RTP アドレスとポート番号を指定できます。アプリケーションは、CiscoMediaTerminal の setRTPParams() でこのオブジェクトを渡すことができます。これらのパラメータは特定のコールだけで有効です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoRTPParams extends java.lang.Object

java.lang.Object

コンストラクタ

CiscoRTPParams (java.net.InetAddress, rtpAddress, int rtpPort)

フィールド

なし

メソッド

表 6-16　CiscoRTPParams のメソッド

インターフェイス	メソッド	説明
java.net.InetAddress	getRTPAddress()	関連付けられたコールの RTP ストリームの受信に使用されるインターネット アドレスを返します。
java.lang.String	getRTPAddressHostName()	関連付けられたコールの RTP ストリームの受信に使用される IP ホスト名を返します。
byte[]	getRTPByteAddress()	RTP ストリームの受信に使用されるインターネット アドレスをバイト形式で返します。
int	getRTPPort()	RTP ストリームの受信に使用される UDP ポートを返します。
java.lang.String	toString()	「IP アドレス / ポート番号」の形式の文字列を返します。クラス java.lang.Object の toString をオーバーライドします。

継承したメソッド

クラス java.lang.Object から
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

CiscoTerminal と CiscoMediaTerminal を参照してください。

CiscoUnregistrationException

CiscoMediaTerminal.unregister メソッドは、登録解除プロセスが失敗すると、この例外をスローします。たとえば、プロバイダーが OUT_OF_SERVICE である場合、または端末がすでに登録解除されている場合、登録は失敗します。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoUnregistrationException extends java.lang.Exception

java.lang.Object

java.lang.Throwable
　java.lang.Exception
　　com.cisco.jtapi.extensions.CiscoUnregistrationException

実装インターフェイス

java.io.Serializable

コンストラクタ

表 6-17　CiscoUnregistrationException のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoUnregistrationException()(java.lang.String description)	なし

フィールド

なし

メソッド

なし

継承したメソッド

クラス java.lang.Throwable から

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

CiscoMediaTerminal.unregister(), Serialized Form を参照してください。

CiscoWideBandMediaCapability

CiscoWideBandMediaCapability オブジェクトは、ワイド バンドで符号化された RTP ストリームのプロパティを指定します。ワイド バンド メディア終端をサポートしているアプリケーションは、CiscoMediaTerminal を登録する際に、このオブジェクトを使用して適切なパケット サイズを指定します。デフォルトのパケット サイズは 10 ミリ秒です。

クラスの履歴

Cisco Unified Communications Manager リリース	説明
7.1x	変更を記録するために履歴表に追加されました。

宣言

public class CiscoWideBandMediaCapability extends CiscoMediaCapability

java.lang.Object

　　com.cisco.jtapi.extensions.CiscoMediaCapability

　　　　com.cisco.jtapi.extensions.CiscoWideBandMediaCapability

コンストラクタ

表 6-18　CiscoWideBandMediaCapability のコンストラクタ

インターフェイス	コンストラクタ	説明
public	CiscoWideBandMediaCapability(int packetsize)	指定されたパケット サイズの CiscoWideBandMediaCapability オブジェクトを構築します。デフォルトのパケット サイズは 10 ミリ秒です。 **パラメータ** • packetsize：RTP パケットのフレームサイズです。

フィールド

表 6-19　CiscoWideBandMedicaCapability のフィールド

インターフェイス	フィールド	説明
static int	FRAMESIZE_TEN_MILLISECOND_PACKET	RTP パケットのフレームサイズ：10 ミリ秒の RTP パケット。

継承したフィールド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から

G711_64K_30_MILLISECONDS, G723_6K_30_MILLISECONDS, G729_30_MILLISECONDS, GSM_80_MILLISECONDS, WIDEBAND_256K_10_MILLISECONDS

メソッド

なし

継承したメソッド

クラス com.cisco.jtapi.extensions.CiscoMediaCapability から

getMaxFramesPerPacket, getPayloadType, isSupported, toString

クラス java.lang.Object から

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

関連資料

「定数フィールド値」（P.F-1）を参照してください。

インターフェイスの階層

次のインターフェイスの階層は com.cisco.jtapi.extensions パッケージ階層に含まれています。

javax.telephony.Address

- com.cisco.jtapi.extensions.CiscoAddress (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)
  - com.cisco.jtapi.extensions.CiscoIntercomAddress
- javax.telephony.callcenter.RouteAddress
  - com.cisco.jtapi.extensions.CiscoRouteAddress

javax.telephony.AddressObserver

- com.cisco.jtapi.extensions.CiscoAddressObserver

javax.telephony.Call

- javax.telephony.callcontrol.CallControlCall
  - com.cisco.jtapi.extensions.CiscoCall (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)
    - com.cisco.jtapi.extensions.CiscoConsultCall

com.cisco.jtapi.extensions.CiscoCallCtlTermConnHeldReversionEv

com.cisco.jtapi.extensions.CiscoConferenceChain

com.cisco.jtapi.extensions.CiscoFeatureReason

com.cisco.jtapi.extensions.CiscoJtapiException

com.cisco.jtapi.extensions.CiscoJtapiProperties

com.cisco.jtapi.extensions.CiscoLocales

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMediaConnectionMode

com.cisco.jtapi.extensions.CiscoMediaEncryptionAlgorithmType

com.cisco.jtapi.extensions.CiscoMediaEncryptionKeyInfo

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMonitorInitiatorInfo

com.cisco.jtapi.extensions.CiscoMonitorTargetInfo

com.cisco.jtapi.extensions.CiscoObjectContainer

- com.cisco.jtapi.extensions.CiscoAddress (javax.telephony.Address も拡張)
  - com.cisco.jtapi.extensions.CiscoIntercomAddress
- com.cisco.jtapi.extensions.CiscoCall (javax.telephony.callcontrol.CallControlCall も拡張)
- com.cisco.jtapi.extensions.CiscoConsultCall
- com.cisco.jtapi.extensions.CiscoCallID
- com.cisco.jtapi.extensions.CiscoConnection (javax.telephony.callcontrol.CallControlConnection も拡張)
- com.cisco.jtapi.extensions.CiscoConnectionID
- com.cisco.jtapi.extensions.CiscoConsultCall
- com.cisco.jtapi.extensions.CiscoIntercomAddress
- com.cisco.jtapi.extensions.CiscoJtapiPeer (javax.telephony.JtapiPeer、com.cisco.services.tracing.TraceModule も拡張)
- com.cisco.jtapi.extensions.CiscoMediaTerminal
- com.cisco.jtapi.extensions.CiscoProvider
- com.cisco.jtapi.extensions.CiscoRouteTerminal
- com.cisco.jtapi.extensions.CiscoTerminal (javax.telephony.Terminal も拡張)
  - com.cisco.jtapi.extensions.CiscoMediaTerminal
  - com.cisco.jtapi.extensions.CiscoRouteTerminal
- com.cisco.jtapi.extensions.CiscoTerminalConnection (javax.telephony.callcontrol.CallControlTerminalConnection も拡張)

com.cisco.jtapi.extensions.CiscoPartyInfo

com.cisco.jtapi.extensions.CiscoProvFeatureID

com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoRecorderInfo

com.cisco.jtapi.extensions.CiscoRTPBitRate

com.cisco.jtapi.extensions.CiscoRTPHandle

com.cisco.jtapi.extensions.CiscoRTPInputProperties

com.cisco.jtapi.extensions.CiscoRTPOutputProperties

com.cisco.jtapi.extensions.CiscoRTPPayload

com.cisco.jtapi.extensions.CiscoSynchronousObserver

com.cisco.jtapi.extensions.CiscoTermConnPrivacyChangedEv

com.cisco.jtapi.extensions.CiscoTermEvFilter

com.cisco.jtapi.extensions.CiscoTerminalProtocol

com.cisco.jtapi.extensions.CiscoTone

com.cisco.jtapi.extensions.CiscoUrlInfo

javax.telephony.Connection

javax.telephony.callcontrol.CallControlConnection

com.cisco.jtapi.extensions.CiscoConnection (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)

javax.telephony.events.Ev

javax.telephony.events.AddrEv

com.cisco.jtapi.extensions.CiscoAddrEv (com.cisco.jtapi.extensions.CiscoEv も拡張)

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張)

com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv

javax.telephony.callcontrol.events.CallCtlEv

javax.telephony.callcontrol.events.CallCtlCallEv（javax.telephony.events.CallEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnEv（javax.telephony.events.ConnEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv

javax.telephony.events.CallEv

javax.telephony.events.CallActiveEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (com.cisco.jtapi.extensions.CiscoCallEv も拡張)

javax.telephony.callcontrol.events.CallCtlCallEv（javax.telephony.callcontrol.events.CallCtlEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnEv（javax.telephony.events.ConnEv も拡張）

javax.telephony.callcontrol.events.CallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv

com.cisco.jtapi.extensions.CiscoCallEv (com.cisco.jtapi.extensions.CiscoEv も拡張)

com.cisco.jtapi.extensions.CiscoCallChangedEv
com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv
com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv
com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv
com.cisco.jtapi.extensions.CiscoConferenceEndEv
com.cisco.jtapi.extensions.CiscoConferenceStartEv
com.cisco.jtapi.extensions.CiscoConsultCallActiveEv
(javax.telephony.events.CallActiveEv も拡張)
com.cisco.jtapi.extensions.CiscoToneChangedEv
com.cisco.jtapi.extensions.CiscoTransferEndEv
com.cisco.jtapi.extensions.CiscoTransferStartEv
javax.telephony.events.ConnEv
javax.telephony.callcontrol.events.CallCtlConnEv
（javax.telephony.callcontrol.events.CallCtlCallEv も拡張）
javax.telephony.callcontrol.events.CallCtlConnOfferedEv
com.cisco.jtapi.extensions.CiscoCallCtlConnOfferedEv
javax.telephony.events.TermConnEv
com.cisco.jtapi.extensions.CiscoTermConnMonitoringEndEv
com.cisco.jtapi.extensions.CiscoTermConnMonitoringStartEv
com.cisco.jtapi.extensions.CiscoTermConnMonitorInitiatorInfoEv
com.cisco.jtapi.extensions.CiscoTermConnMonitorTargetInfoEv
com.cisco.jtapi.extensions.CiscoTermConnRecordingEndEv
com.cisco.jtapi.extensions.CiscoTermConnRecordingStartEv
com.cisco.jtapi.extensions.CiscoTermConnRecordingTargetInfoEv
com.cisco.jtapi.extensions.CiscoTermConnSelectChangedEv
com.cisco.jtapi.extensions.CiscoEv
com.cisco.jtapi.extensions.CiscoAddrActivatedEv
com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv
com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
com.cisco.jtapi.extensions.CiscoAddrCreatedEv
com.cisco.jtapi.extensions.CiscoAddrEv (javax.telephony.events.AddrEv も拡張)
com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
com.cisco.jtapi.extensions.CiscoAddrInServiceEv
com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv
com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv
com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv
(com.cisco.jtapi.extensions.CiscoAddrEv、com.cisco.jtapi.extensions.CiscoOutOfServiceEv
も拡張)

com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv
(com.cisco.jtapi.extensions.CiscoAddrEv も拡張)

com.cisco.jtapi.extensions.CiscoAddressRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallEv (javax.telephony.events.CallEv も拡張)

com.cisco.jtapi.extensions.CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (javax.telephony.events.CiscoCallEv も拡張)

com.cisco.jtapi.extensions.CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferStartEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv (javax.telephony.events.CallActiveEv、com.cisco.jtapi.extensions.CiscoCallEv も拡張)

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoOutOfServiceEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv (com.cisco.jtapi.extensions.CiscoAddrEv も拡張)

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv (com.cisco.jtapi.extensions.CiscoTermEv も拡張)

com.cisco.jtapi.extensions.CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvFeatureEv (javax.telephony.events.ProvEv も拡張)

com.cisco.jtapi.extensions.CiscoAddrActivatedEv
com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv
com.cisco.jtapi.extensions.CiscoAddrCreatedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoProvFeatureEv
    com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
    com.cisco.jtapi.extensions.CiscoTermActivatedEv
    com.cisco.jtapi.extensions.CiscoTermCreatedEv
    com.cisco.jtapi.extensions.CiscoTermRemovedEv
    com.cisco.jtapi.extensions.CiscoTermRestrictedEv
    com.cisco.jtapi.extensions.CiscoProvFeatureEv
    com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
    com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
com.cisco.jtapi.extensions.CiscoTermActivatedEv
com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
com.cisco.jtapi.extensions.CiscoTermCreatedEv
com.cisco.jtapi.extensions.CiscoTermDataEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermEvFilter (javax.telephony.events.TermEv も拡張)

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoRTPInputKeyEv

com.cisco.jtapi.extensions.CiscoRTPInputStartedEv

com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv

com.cisco.jtapi.extensions.CiscoTermButtonPressedEv

com.cisco.jtapi.extensions.CiscoTermDataEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv(com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張)

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv
(com.cisco.jtapi.extensions.CiscoOutOfServiceEv、com.cisco.jtapi.extensions.CiscoTermEv も拡張)

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermRemovedEv

com.cisco.jtapi.extensions.CiscoTermRestrictedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferStartEv

javax.telephony.events.ProvEv

com.cisco.jtapi.extensions.CiscoProvEv （com.cisco.jtapi.extensions.CiscoEv も拡張）

com.cisco.jtapi.extensions.CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv
com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv
com.cisco.jtapi.extensions.CiscoAddrCreatedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedEv
com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoProvFeatureEv
com.cisco.jtapi.extensions.CiscoProvCallParkEv
com.cisco.jtapi.extensions.CiscoRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedEv
com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv
com.cisco.jtapi.extensions.CiscoTermActivatedEv
com.cisco.jtapi.extensions.CiscoTermCreatedEv
com.cisco.jtapi.extensions.CiscoTermRemovedEv
com.cisco.jtapi.extensions.CiscoTermRestrictedEv

javax.telephony.events.TermEv

com.cisco.jtapi.extensions.CiscoTermEv (com.cisco.jtapi.extensions.CiscoEv も拡張)
com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv
com.cisco.jtapi.extensions.CiscoRTPInputKeyEv
com.cisco.jtapi.extensions.CiscoRTPInputStartedEv
com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv
com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv
com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv
com.cisco.jtapi.extensions.CiscoRTPOutputStoppedEv
com.cisco.jtapi.extensions.CiscoTermButtonPressedEv
com.cisco.jtapi.extensions.CiscoTermDataEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv
com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv
com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv
com.cisco.jtapi.extensions.CiscoTermInServiceEv
com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv
(com.cisco.jtapi.extensions.CiscoOutOfServiceEv も拡張)
com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotEv

javax.telephony.JtapiPeer

com.cisco.jtapi.extensions.CiscoJtapiPeer (com.cisco.jtapi.extensions.CiscoObjectContainer, com.cisco.services.tracing.TraceModule も拡張)

javax.telephony.Provider

com.cisco.jtapi.extensions.CiscoProvider (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)

javax.telephony.capabilities.ProviderCapabilities

com.cisco.jtapi.extensions.CiscoProviderCapabilities

javax.telephony.ProviderObserver

com.cisco.jtapi.extensions.CiscoProviderObserver

javax.telephony.callcenter.RouteSession

com.cisco.jtapi.extensions.CiscoRouteSession

javax.telephony.callcenter.events.RouteSessionEvent

javax.telephony.callcenter.events.RouteEvent

com.cisco.jtapi.extensions.CiscoRouteEvent

javax.telephony.callcenter.events.RouteUsedEvent

com.cisco.jtapi.extensions.CiscoRouteUsedEvent

javax.telephony.Terminal

com.cisco.jtapi.extensions.CiscoTerminal (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)

com.cisco.jtapi.extensions.CiscoMediaTerminal

com.cisco.jtapi.extensions.CiscoRouteTerminal

javax.telephony.TerminalConnection

javax.telephony.callcontrol.CallControlTerminalConnection

com.cisco.jtapi.extensions.CiscoTerminalConnection (com.cisco.jtapi.extensions.CiscoObjectContainer も拡張)

javax.telephony.TerminalObserver

com.cisco.jtapi.extensions.CiscoTerminalObserver

com.cisco.services.tracing.TraceModule

com.cisco.jtapi.extensions.CiscoJtapiPeer (com.cisco.jtapi.extensions.CiscoObjectContainer, javax.telephony.JtapiPeer も拡張)

CiscoAddrActivatedEv

アドレスが制御され、「restriction（制限）」状態が active に変更された場合、CiscoAddrActivatedEv イベントがアプリケーションに送信されます。アプリケーションは、アドレスまたは関連付けられた端末が制御リストに含まれている場合に、このイベントを受信します。該当アドレスにオブザーバが存在

する場合、アプリケーションは CiscoAddrInServiceEv を受信します。オブザーバが存在しない場合、アプリケーションはオブザーバの追加を試みることが可能で、アドレスはイン サービス状態になります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrActivatedEv extends CiscoProvEv

フィールド

表 6-20　CiscoAddrActivatedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-21　CiscoAddrActivatedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	有効化されたアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.ProvEv から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、定数フィールド値を参照してください。

CiscoCallCtlConnOfferedEv インターフェイスは CallCtlConnOfferedEv インターフェイスを拡張して、アプリケーションが発信側端末の IP アドレスを取得できるようにします。すべての発信側デバイスの IP アドレス情報が参照可能とは限りません。戻り値が 0（または null）の場合、情報が取得可能でないことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlConnEv, javax.telephony.callcontrol.events.CallCtlConnOfferedEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.events.CallEv, javax.telephony.events.ConnEv, javax.telephony.events.Ev

宣言

public interface CiscoCallCtlConnOfferedEv extends javax.telephony.callcontrol.events.CallCtlConnOfferedEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.callcontrol.events.CallCtlConnOfferedEv から
なし

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から
CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-22　CiscoCallCtlConnOfferedEv のメソッド

インターフェイス	メソッド	説明
java.net.InetAddress	getCallingPartyIpAddr()	発信側の IP アドレスか、IP アドレスが取得できない場合は 0（または null）を返します。

継承したメソッド

インターフェイス javax.telephony.callcontrol.events.CallCtlCallEv から
getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から
getCallControlCause

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ConnEv から
getConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

なし

CiscoAddrActivatedOnTerminalEv

CiscoAddrActivatedOnTerminalEv イベントは、共用回線が有効になるか、または共用回線を持つ端末が有効になると送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrActivatedOnTerminalEv extends CiscoProvEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-23　CiscoAddrActivatedOnTerminalEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	端末で制限解除とマークされているアドレスを返します。
javax.telephony.Terminal	getTerminal()	アドレスが有効になっている（制限解除とマークされている）端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、定数フィールド値を参照してください。

CiscoAddrAddedToTerminalEv

次の場合に、CiscoAddrAddedToTerminalEv が送信されます。

- 共用回線が含まれている制御リストにユーザが端末を追加すると、このイベントがアプリケーションに送信されます。ユーザが制御リストにアドレスを持っている場合、制御リスト内にあるのと同じアドレスを持つ新しい端末を追加すると、このイベントが送信されます。
- エクステンション モビリティ（EM）ユーザが共用回線を持つプロファイルで端末にログインすると、このイベントによって新しい端末が既存のアドレスに追加されたことが通知されます。
- 新しい共用回線がユーザの制御リストに追加されると、このイベントがアプリケーションに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrAddedToTerminalEv extends CiscoProvEv

フィールド

表 6-24　CiscoAddrAddedToTerminalEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-25　CiscoAddrAddedToTerminalEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	新しい端末が追加されるアドレスを返します。
javax.telephony.Terminal	getTerminal()	アドレスに追加される端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、定数フィールド値を参照してください。

CiscoAddrAutoAcceptStatusChangedEv

CiscoAddrAutoAcceptStatusChangedEv は、端末のアドレスの AutoAccept ステータスが変更されるたびにアプリケーションに送信されます。アドレスに複数の端末がある場合、このイベントは個々の端末のアドレスの AutoAccept ステータスごとに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrAutoAcceptStatusChangedEv extends CiscoAddrEv

フィールド

表 6-26　CiscoAddrAutoAcceptStatusChangedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUTUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_R_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-27　CiscoAddrAutoAcceptStatusChangedEv のメソッド

インターフェイス	メソッド	説明
int	getAutoAcceptStatus()	端末上のアドレスの AutoAccept ステータスを返します。CiscoAddress.AUTOACCEPT_OFF または CiscoAddress.AUTOACCEPT_ON を返します。
CiscoTerminal	getTerminal()	このアドレスの AutoAccept ステータスが変更される端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

getAutoAcceptStatus および CiscoAddress.getAutoAcceptStatus(Terminal terminal) を参照してください。

CiscoAddrCreatedEv

CiscoAddrCreatedEv イベントは、アドレスがプロバイダー ドメインに追加されるときに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrCreatedEv extends CiscoProvEv

フィールド

表 6-28　CiscoAddrCreatedEv のフィールド

インターフェイス	フィールド
ID	static final int ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-29　CiscoAddrCreatedEv のメソッド

インターフェイス	メソッド	説明
getAddress	javax.telephony.Address getAddress()	プロバイダー ドメインに追加されたアドレスを返します。プロバイダー ドメインに追加されるアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoAddress

CiscoAddress インターフェイスは、Cisco Unified Communications Manager の機能を追加することによって、アドレス インターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	電話だけをサポートする Message Waiting Indication（MWI；メッセージ受信通知）機能が拡張され、音声とファックスのメッセージ数が追加されました。

スーパーインターフェイス

javax.telephony.Address, CiscoObjectContainer

サブインターフェイス

CiscoIntercomAddress

フィールド

表 6-30　CiscoAddress のフィールド

インターフェイス	フィールド	説明
Static int	APPLICATION_CONTROLLED_RECORDING	アプリケーションによって制御される録音は、アドレスに設定されます。
Static int	AUTO_RECORDING	自動録音は、アドレスに設定されます。
Static int	AUTOACCEPT_OFF	AutoAccept はオフです。
Static int	AUTOACCEPT_ON	AutoAccept はオンです。
Static int	AUTOANSWER_OFF	AutoAnswer はオフです。
Static int	AUTOANSWER_UNKNOWN	AutoAnswer のステータスは不明です。
Static int	AUTOANSWER_WITHHEADSET	ヘッドセットを使用した AutoAnswer は許可されます。
static int	AUTOANSWER_WITHSPEAKERSET	スピーカー セットを使用した AutoAnswer は許可されます。
static int	EXTERNAL	これは有効な名前の外部アドレスを表します。
static int	EXTERNAL_UNKNOWN	これは不明な名前の外部アドレスを表します。
static int	IN_SERVICE	アドレスがイン サービスです。
static int	INTERNAL	これは内部アドレスです。
static int	MONITORING_TARGET	これはモニタリングのターゲットまたはエージェントのあるアドレスを表します。
static int	NO_RECORDING	アドレスに対して録音がオフになっています。
static int	OUT_OF_SERVICE	アドレスがアウトオブサービスです。
static int	RINGER_DEFAULT	呼び出し音のステータスを設定された値に変更します。

表 6-30　CiscoAddress のフィールド

インターフェイス	フィールド	説明
static int	RINGER_DISABLE	アドレスの呼び出し音を無効にします。
static int	RINGER_ENABLE	アドレスの呼び出し音を有効にします。
static int	UNKNOWN	これは不明な名前のアドレスを表します。

メソッド

表 6-31　CiscoAddress のメソッド

インターフェイス	メソッド	説明
void	clearCallConnections()	このインターフェイスを使用して、アドレス上からファントム コールをクリアします。 **例外** javax.telephony.PrivilegeViolationException— このインターフェイスを使用して、アドレス上からファントム コールをクリアします。
CiscoAddressCallInfo	getAddressCallInfo (javax.telephony.Terminal terminal)	このインターフェイスを使用して、端末に存在するコールに関する情報を取得します。
int	getAutoAcceptStatus(javax.telephony.Terminal terminal)	端末上のアドレスの AutoAccept ステータスを返します。 **例外** javax.telephony.PlatformException, javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException 端末 繧Ãアドレスの AutoAccept ステータスを返します。次のいずれかの定数を返します。 • CiscoAddress.AUTOACCEPT_OFF • CiscoAddress.AUTOACCEPT_ON **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE getState() == IN_SERVICE **パラメータ** • terminal：AutoAccepts の端末

表 6-31　CiscoAddress のメソッド （続き）

インターフェイス	メソッド	説明
int	getAutoAnswerStatus(javax.telephony.Terminal term)	このインターフェイスは、指定された端末のこのアドレスの AutoAnswer ステータスを返します。 **例外** javax.telephony.PlatformException, javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException 返される値が AUTOANSWER_OFF の場合、AutoAnswer が無効であることを意味します。返される値が AUTOANSWER_WITHHEADSET の場合、HEADSET で AutoAnswer が有効であることを意味します。返される値が AUTOANSWER_WITHSPEAKERSET の場合、SPEAKERSET で AutoAnswer が有効であることを意味します。返される値は AUTOANSWER_UNKNOWN の場合、AutoAnswer ステータスが UNKNOWN であることを意味します。 **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE getState() == IN_SERVICE **パラメータ** • term：AutoAnswer がオンになっている端末。 次のいずれかの値を返します。 • CiscoAddress.AUTOANSWER_OFF • CiscoAddress.AUTOANSWER_ WITHHEADSET • CiscoAddress.AUTOANSWER_ WITHSPEAKERSET • CiscoAddress.AUTOANSWER_UNKNOWN **例外** javax.telephony.InvalidStateException：プロバイダーまたはアドレスが「イン サービス」ではありません。 javax.telephony.PlatformException：アドレスが Terminal term にない場合 javax.telephony.MethodNotSupportedException：アドレスが外部アドレスの場合
javax.telephony.Terminal[]	getInServiceAddrTerminals()	このインターフェイスを使用して、イン サービスである共用回線を検出します。共用回線では、複数の端末で同じアドレスが現れます。 戻り値：Terminal[] — このアドレスがイン サービス状態にある端末の配列。
java.lang.String	getPartition()	アドレスに関連付けられたパーティションを返します。

表 6-31　CiscoAddress のメソッド （続き）

インターフェイス	メソッド	説明
int	getRecordingConfig(javax.telephony.Terminal term)	このアドレスに設定された録音タイプを返します。 **例外** javax.telephony.PlatformException,javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException **戻り値** • int：このアドレスに設定された録音タイプを返します。 • CiscoAddess.NO_RECORDING：コールを録音できません。 • CiscoAddress.AUTO_RECORDING：Unified CM はこのアドレスで応答したすべてのコールを録音します。 • CiscoAddress.APPLICATION_CONTROLLED_RECORDING：アプリケーションが録音を開始したときだけコールが録音されます。 **例外** javax.telephony.InvalidStateException：プロバイダーまたはアドレスが「イン サービス」ではありません。 javax.telephony.PlatformException：アドレスが Terminal term にない場合 javax.telephony.MethodNotSupportedException：アドレスが外部アドレスの場合
int	getRegistrationState()	**推奨されません。** このメソッドは getState() メソッドに置き換えられました。次の定数のいずれかの可能性があるこのアドレスの状態を返します。 • CiscoAddress.OUT_OF_SERVICE • CiscoAddress.IN_SERVICE
javax.telephony.Terminal[]	getRestrictedAddrTerminals()	このアドレスが制限されている端末の配列を返します。共用回線の場合、端末上のいくつかの回線が制限されている可能性があります。 アプリケーションは、制限されたアドレスのコール イベントは受信できません。制限されたアドレスが他の制御端末とのコールに関わっている場合、制限されたアドレス用の接続が作成されますが、制限されたアドレス用の TerminalConnection はありません。 戻り値：Terminal[] — このアドレスが制限されている端末の配列。制限されている端末がない場合、このメソッドは null を返します。

表 6-31　CiscoAddress のメソッド （続き）

インターフェイス	メソッド	説明
int	getState()	このアドレスの状態を返します。状態を表す定数は次のとおりです。 • CiscoAddress.OUT_OF_SERVICE • CiscoAddress.IN_SERVICE
int	getType()	次のアドレス定数を返します。 • CiscoAddress.INTERNAL • CiscoAddress.EXTERNAL • CiscoAddress.EXTERNAL_UNKNOWN • CiscoAddress.UNKNOWN • CiscoAddress.MONITORING_TARGET
boolean	isRestricted(javax.telephony.Terminal terminal)	このメソッドは、端末上のこのアドレスが制限されている場合、true を返し、制限されていない場合、false を返します。

表 6-31　CiscoAddress のメソッド（続き）

インターフェイス	メソッド	説明
void	setAutoAcceptStatus(int autoAcceptStatus, javax.telephony.Terminal terminal)	このメソッドでは、アプリケーションが CiscoMediaTerminal や CiscoRouteTerminal 上のアドレスに対して AutoAccept を有効にできるようになります。 **例外** javax.telephony.PlatformException, javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException このメソッドでは、アプリケーションが CiscoMediaTerminal や CiscoRouteTerminal 上のアドレスに対して AutoAccept を有効にできるようになります。 CiscoMediaTerminal または CiscoRouteTerminal 以外の CiscoTerminal 上のアドレスの AutoAccept は常にオンになります。パラメータとして渡された端末が CiscoMediaTerminal や CiscoRouteTerminal でない場合、このメソッドは例外をスローします。 CiscoTerminal とアドレスを共用している CiscoMediaTerminal の場合は、CiscoMediaTerminal で AutoAccept を有効にすることを推奨します。 **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE getState() == IN_SERVICE **事後条件** 自動応答のステータスを有効または無効に設定します。 **パラメータ** • autoAcceptStatus：CiscoAddress.AUTOACCEPT_OFF または CiscoAddress.AUTOACCEPT_ON のいずれかにできます。autoAcceptStatus が AUTOACCEPT_ON の場合、端末のアドレスに対して AutoAccept が有効にされます。autoAcceptStatus が AUTOACCEPT_OFF の場合、端末のアドレスに対して AutoAccept が無効にされます。 • terminal：AutoAccept が有効にされる端末。 **例外** javax.telephony.InvalidStateException：プロバイダーまたはアドレスが「イン サービス」ではありません。 javax.telephony.PlatformException：この端末にはこのアドレスがありません。 javax.telephony.MethodNotSupportedException：端末が CiscoMediaTerminal または CiscoRouteTerminal でない場合。

表 6-31　CiscoAddress のメソッド（続き）

インターフェイス	メソッド	説明
void	setMessageWaiting(java.lang.String destination, boolean enable)	destination で指定されたアドレスのメッセージ受信インジケータを有効にするか無効にするかを指定します。enable が true の場合は、メッセージ受信が有効になります（すでに有効になっている場合はそのまま）。enable が false の場合は、メッセージ受信が無効になります（すでに無効になっている場合はそのまま）。 **例外** javax.telephony.MethodNotSupportedException, javax.telephony.InvalidStateException,javax.telephony.PrivilegeViolationException **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE **事後条件** 有効/無効のステータスに応じて、メッセージ受信インジケータを有効または無効にします。 **（注）** この実装は現在、CallControlAddress で次のように指定されている事後条件を実現しません。 this.getMessageWaiting() == enable このアドレスに CallCtlAddrMessageWaitingEv が配信されます。 **パラメータ** • destination：メッセージ受信インジケータを有効または無効にする DN/アドレス。 • enable：メッセージ受信を有効にする場合は true、無効にする場合は false。 **例外** • javax.telephony.MethodNotSupportedException：このメソッドは指定された実装でサポートされていません。 javax.telephony.InvalidStateException **（注）** プロバイダーが「イン サービス」ではありません。 javax.telephony.PrivilegeViolationException **（注）** プロバイダーのユーザに、この宛先のメッセージ待ちインジケータを起動する権限がありません。

表 6-31　CiscoAddress のメソッド（続き）

インターフェイス	メソッド	説明
void	setRingerStatus(int status)	このアドレスの呼び出し音ステータスを変更します。 **例外** javax.telephony.MethodNotSupportedException, javax.telephony.InvalidStateException, javax.telephony.InvalidArgumentException 次に示す定数のうちの 1 つを受け入れます。 • CiscoAddress.RINGER_DEFAULT • CiscoAddress.RINGER_DISABLE • CiscoAddress.RINGER_ENABLE
void	setMessageSummary (boolean enable, boolean voiceCounts, int totalNewVoiceMsgs, int totalOldVoiceMsgs, boolean highPriorityVoiceCounts, int newHighPriorityVoiceMsgs, int oldHighPriorityVoiceMsgs, boolean faxCounts, int totalNewFaxMsgs, int totalOldFaxMsgs, boolean highPriorityFaxCounts, int newHighPriorityFaxMsgs, int oldHighPriorityFaxMsgs)	このインターフェイスを使用して、メッセージ受信インジケータと音声/ファックスのメッセージ受信数を設定します。enable が true の場合は、メッセージ受信が有効になります（すでに有効になっている場合はそのまま）。enable が false の場合は、メッセージ受信が無効になります（すでに無効になっている場合はそのまま）。 **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE **事後条件** メッセージ受信インジケータを有効または無効にして、メッセージ受信数を設定します。

表 6-31　CiscoAddress のメソッド（続き）

インターフェイス	メソッド	説明
		パラメータ • enable：メッセージ受信を有効にする場合は true、無効にする場合は false。 • voiceCounts：音声メッセージ数が指定されているかどうかを示します。 • totalNewVoiceMsgs：新しい音声メッセージ受信の合計数を指定します。 • totalOldVoiceMsgs：古い音声メッセージ受信の合計数を指定します。 • highPriorityVoiceCounts：優先度の高い音声メッセージ数 ™ 指定されているかどうかを示します。 • newHighPriorityVoiceMsgs：優先度の高い新しい音声メッセージ受信の合計数を指定します。 • oldHighPriorityVoiceMsgs：優先度の高い古い音声メッセージ受信の合計数を指定します。 • faxCounts：ファックス メッセージ数が指定されているかどうかを示します。 • totalNewFaxMsgs：新しいファックス メッセージ受信の合計数を指定します。 • totalOldFaxMsgs：古いファックス メッセージ受信の合計数を指定します。 • highPriorityFaxCounts：優先度の高いファックス メッセージ数が指定されているかどうかを示します。 • newHighPriorityFaxMsgs：優先度の高い新しいファックス メッセージ受信の合計数を指定します。 • oldHighPriorityFaxMsgs：優先度の高い古いファックス メッセージ受信の合計数を指定します。 例外 javax.telephony.MethodNotSupportedException：このメソッドは指定された実装でサポートされていません。 javax.telephony.InvalidStateException：プロバイダーが「インサービス」ではありません。 javax.telephony.PrivilegeViolationException：ユーザに、この宛先のメッセージ受信インジケータを設定する権限がありません。

表 6-31　CiscoAddress のメソッド（続き）

インターフェイス	メソッド	説明
void	setMessageSummary(java.lang.String destination, boolean enable, boolean voiceCounts, int totalNewVoiceMsgs, int totalOldVoiceMsgs, boolean highPriorityVoiceCounts, int newHighPriorityVoiceMsgs, int oldHighPriorityVoiceMsgs, boolean faxCounts, int totalNewFaxMsgs, int totalOldFaxMsgs, boolean highPriorityFaxCounts, int newHighPriorityFaxMsgs, int oldHighPriorityFaxMsgs)	このインターフェイスを使用して、宛先によって指定されたアドレスに対するメッセージ受信インジケータと音声/ファックスのメッセージ受信数を設定します。 **事前条件** (this.getProvider()).getState() == Provider.IN_SERVICE **事後条件** メッセージ受信インジケータを有効または無効にして、メッセージ受信数を設定します。 **パラメータ** • destination：メッセージ待機インジケータを有効または無効にする DN/アドレス。 • enable：メッセージ受信を有効にする場合は true、無効にする場合は false。 • voiceCounts：音声メッセージ数が指定されているかどうかを示します。 • totalNewVoiceMsgs：新しい音声メッセージ受信の合計数を指定します。 • totalOldVoiceMsgs：古い音声メッセージ受信の合計数を指定します。 • highPriorityVoiceCounts：優先度の高い音声メッセージ数が指定されているかどうかを示します。 • newHighPriorityVoiceMsgs：優先度の高い新しい音声メッセージ受信の合計数を指定します。 • oldHighPriorityVoiceMsgs：優先度の高い古い音声メッセージ受信の合計数を指定します。 • faxCounts：ファックス メッセージ数が指定されているかどうかを示します。 • totalNewFaxMsgs：新しいファックス メッセージ受信の合計数を指定します。 • totalOldFaxMsgs：古いファックス メッセージ受信の合計数を指定します。 • highPriorityFaxCounts：優先度の高いファックス メッセージ数が指定されているかどうかを示します。 • newHighPriorityFaxMsgs：優先度の高い新しいファックス メッセージ受信の合計数を指定します。 • oldHighPriorityFaxMsgs：優先度の高い古いファックス メッセージ受信の合計数を指定します。

継承したメソッド

インターフェイス javax.telephony.Address から
addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoAddressObserver

アプリケーションは、Address.addObserver メソッドを使用してアドレスを監視する際に、このインターフェイスを実装して CiscoAddrInServiceEv や CiscoAddrOutOfServiceEv などの CiscoAddrEv イベントを受信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.AddressObserver

宣言

public interface CiscoAddressObserver extends javax.telephony.AddressObserver

フィールド

なし

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.AddressObserver から
addressChangedEvent

関連資料

詳細については、CiscoAddrInServiceEv と CiscoAddrOutOfServiceEv を参照してください。

CiscoAddrEv

JTAPI のコアである javax.telephony.events.AddrEv インターフェイスを拡張する CiscoAddrEv インターフェイスは、Cisco によって拡張されたすべての JTAPI アドレス イベントの基本インターフェイスになります。このパッケージのアドレス関連イベントはすべて、直接的または間接的にこのインターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoEv, javax.telephony.events.Ev

サブインターフェイス

CiscoAddrAutoAcceptStatusChangedEv, CiscoAddrInServiceEv, CiscoAddrIntercomInfoChangedEv, CiscoAddrIntercomInfoRestorationFailedEv, CiscoAddrOutOfServiceEv, CiscoAddrRecordingConfigChangedEv

宣言

public interface CiscoAddrEv extends CiscoEv, javax.telephony.events.AddrEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、javax.telephony.events.AddrEv を参照してください。

CiscoAddrCreatedEv

アドレス イベントのフィルタを設定するためにアプリケーションに提供された CiscoAddrEvFilter。アプリケーションは次の API を使用して、アドレスに対するイベント通知を受信するためのフィルタを有効または無効に設定したり、またはフィルタに設定された値を確認したりできます。アプリケーションはこのフィルタを有効にすることができます。新しいイベントを受信する場合（CiscoAddrParkStatusEv）、デフォルトではその他のイベントに対するフィルタ値が true です。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	パーク モニタリングおよび Assisted DPark サポート機能に対してこのイベントが追加されました。

フィールド

なし

メソッド

表 6-32　CiscoAddrEvFilter のメソッド

インターフェイス	メソッド	説明
boolean	getCiscoAddrParkStatusEvFilter()	アプリケーションはこの API を起動して、CiscoAddrParkStatusEv に対するフィルタのステータスを確認できます。返されるデフォルト値は false です。
Void	setCiscoAddrParkStatusEvFilter (Boolean filterValue)	アプリケーションはこの API を起動して、CiscoAddrParkStatusEv に対するフィルタのステータスを設定できます。
boolean	getCiscoAddrIntercomInfo ChangedEvFilter()	アプリケーションはこの API を起動して、CiscoAddrIntercomInfoChangedEv に対するフィルタのステータスを確認できます。デフォルト値は true です。
void	setCiscoAddrIntercomInfo ChangedEvFilter(boolean filter value)	アプリケーションはこの API を起動して、CiscoAddrIntercomInfoChangedEv に対するフィルタのステータスを設定できます。
boolean	getCiscoAddrIntercomInfo RestorationFailedEvFilter()	アプリケーションはこの API を起動して、CiscoAddrIntercomInfoRestorationFailedEv に対するフィルタのステータスを確認できます。デフォルト値は true です。
void	setCiscoAddrIntercomInfoRestoration FailedEvFilter(boolean filter value)	アプリケーションはこの API を起動して、CiscoAddrIntercomInfoRestorationFailedEv に対するフィルタのステータスを設定できます。

表 6-32　CiscoAddrEvFilter のメソッド （続き）

インターフェイス	メソッド	説明
boolean	getCiscoAddrRecordingConfig ChangedEvFilter()	アプリケーションはこの API を起動して、CiscoAddrRecordingConfigChangedEv に対するフィルタのステータスを取得できます。デフォルト値は true です。
void	setCiscoAddrRecordingConfig ChangedEvFilter(boolean filter value)	アプリケーションはこの API を起動して、CiscoAddrRecordingConfigChangedEv に対するフィルタの値を設定できます。

継承したメソッド

なし

パラメータ

set メソッドはブール値をパラメータとして使用します。

値の範囲

get メソッドはブール値（true または false）を返します。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrInServiceEv

CiscoAddrInServiceEv はアドレスが現在、IN_SERVICE であることを示します。共用回線（複数の端末で同じアドレスが表示される）では、アプリケーションがすべての端末に対して複数の CiscoAddressInService イベントを受け取ることがあります。アプリケーションはこのインターフェイスを使用して、アドレス（または共用回線）が IN_SERVICE になる端末を検出できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrInServiceEv extends CiscoAddrEv

フィールド

表 6-33　CiscoAddrInService のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-34　CiscoAddrInService のメソッド

インターフェイス	メソッド	説明
getTerminal	CiscoTerminal getTerminal()	このアドレスが IN_SERVICE になる端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、CiscoAddress.getInServiceAddrTerminals() と「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrIntercomInfoChangedEv

CiscoIntercomAddress のターゲットの DN またはインターコム ターゲット ラベルが変更されると、CiscoAddrIntercomInfoChangedEv イベントがアプリケーションに送信されます。このイベントは、CiscoIntercomAddress に追加されたすべてのアプリケーション オブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrIntercomInfoChangedEv extends CiscoAddrEv

フィールド

表 6-35　CiscoAddrIntercomInfoChangedEv のフィールド

インターフェイス	フィールド
Static Int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-36　CiscoAddrIntercomInfoChangedEv のメソッド

インターフェイス	メソッド	説明
getIntercomAddress	getIntercomAddress()	情報が変更されたインターコムのアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、CiscoAddrEv と「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrIntercomInfoRestorationFailedEv

フェールオーバーまたはフェールバックの発生時に、アプリケーションが設定したインターコム アドレスのインターコム ターゲット DN または CiscoIntercomAddress のインターコム ターゲット ラベルを JTAPI が復元できなかった場合に、CiscoAddrIntercomInfoRestorationFailedEv イベントがアプリケーションに送信されます。このイベントは、インターコム ターゲット DN またはインターコム ターゲット ラベルを設定したアプリケーションのためのアプリケーション オブザーバで提供されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrIntercomInfoRestorationFailedEv extends CiscoAddrEv

フィールド

表 6-37　CiscoAddrIntercomInfoRestorationFailedEv のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE,

CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-38　CiscoAddrIntercomInfoRestorationFailedEv のメソッド

インターフェイス	メソッド	説明
CiscoIntercomAddress	getIntercomAddress()	このインターフェイスは、情報の復元が失敗したインターコムの Cisco IntercomAddress を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）と CiscoAddrEv を参照してください。

CiscoAddrOutOfServiceEv

CiscoAddrOutOfServiceEv イベントは、アドレスが OUT_OF_SERVICE になったことをアプリケーションに通知します。共用回線（複数の端末で同じアドレスが表示される）では、アプリケーションがすべての端末に対して複数の CiscoAddrOutOfServiceEv イベントを受け取ることがあります。アプリケーションはこのインターフェイスを使用して、アドレス（または共用回線）が OUT_OF_SERVICE になる端末を検出できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, CiscoOutOfServiceEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrOutOfServiceEv extends CiscoAddrEv, CiscoOutOfServiceEv

フィールド

表 6-39　CiscoAddrOutOfServiceEv のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス com.cisco.jtapi.extensions.CiscoOutOfServiceEv から
CAUSE_CALLMANAGER_FAILURE, CAUSE_CTIMANAGER_FAILURE, CAUSE_DEVICE_FAILURE, CAUSE_DEVICE_RESTRICTED, CAUSE_DEVICE_UNREGISTERED, CAUSE_LINE_RESTRICTED, CAUSE_NOCALLMANAGER_AVAILABLE, CAUSE_REHOME_TO_HIGHER_PRIORITY_CM, CAUSE_REHOMING_FAILURE

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-40　CiscoAddrOutOfServiceEv のメソッド

インターフェイス	メソッド	説明
CiscoTerminal	getTerminal()	このアドレスが OUT_OF_SERVICE になる端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）と CiscoAddress.getInServiceAddrTerminals() を参照してください。

CiscoAddrParkStatusEv

Cisco Unified IP Phone を使用してコールをパークする場合、JTAPI はこのイベントを使用してパークの状態を報告します。これは、パークが起動されたアドレスにアドレスのオブザーバが追加されたすべてのアプリケーションに報告されます。このイベントは、Cisco Unified IP Phone からパークが起動された場合にだけ配信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	パーク モニタリングおよび Assisted DPark 機能のためのインターフェイスが追加されました。

宣言

public interface CiscoAddrParkStatusEv extends CiscoAddrEv

フィールド

表 6-41　CiscoAddrParkStatusEv のフィールド

インターフェイス	フィールド	説明
static int	PARKED	コールがパークされるときのパークのステータス。
static int	REMINDER	パーク モニタリング復帰タイマーの期限が切れたときのパークのステータス。
static int	RETRIEVED	パークされたコールがパーク元または第三者によって取得されるときのパークのステータス。
static int	FORWARDED	Park Monitoring Forward-No-retrieve タイマーの期限が切れて、パークされたコールが転送されるときのパークのステータス。
static int	ABANDONED	パークされたコールが接続解除されるときのパークのステータス。

継承したフィールド

なし

メソッド

表 6-42　CiscoAddrParkStatusEv のメソッド

インターフェイス	メソッド	説明
int	getParkState()	パークされたコールの現在のパーク状態を返します。
int	getTransactionID()	パークされた特定のコールに固有の ID を返します。パークされた同じコールに対する複数のパーク状態のトランザクション ID は同じです。

表 6-42　CiscoAddrParkStatusEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCiscoCallID()	CiscoCallID を返します。
String	getParkDN()	コールがパークされている DN を返します。
String	getParkDNPartition()	パーク DN のパーティションを返します。
String	getParkedParty()	パークされた通話者の DN を返します。
String	getParkedPartyPartition()	パークされた通話者のパーティションを返します。
Terminal	getTerminal()	このイベントが配信されたアドレスの端末を返します。

値の範囲

フィールドの値は次のとおりです。

- PARKED: 2
- REMINDER: 3
- RETRIEVED: 4
- ABANDONED: 5
- FORWARDED: 6

関連資料

詳細については、定数フィールド値を参照してください。

CiscoAddrRecordingConfigChangedEv

CiscoAddrRecordingConfigChangedEv イベントは、アドレスの録音設定が変更されると、アドレスのオブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.AddrEv, CiscoAddrEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoAddrRecordingConfigChangedEv extends CiscoAddrEv

フィールド

表 6-43　CiscoAddrRecordingConfigChangedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-44　CiscoAddrRecordingConfigChangedEv のメソッド

インターフェイス	メソッド	説明
Int	getRecordingConfig()	このアドレスの新しい録音設定を返します。値は次のいずれかになります。 • CiscoAddress.NO_RECORDING • CiscoAddress.AUTO_RECORDING • CiscoAddress.APPLICATION_CONTROLLED_RECORDING
javax.telephony.Terminal	getTerminal()	録音設定が変更された端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.AddrEv から
getAddress

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）と CiscoAddrEv を参照してください。

CiscoAddrRemovedEv

JTAPI は、アドレスがプロバイダー ドメインから削除されるときに CiscoAddrRemovedEv イベントを送信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrRemovedEv extends CiscoProvEv

フィールド

表 6-45　CiscoAddrRemovedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-46　CiscoAddrRemovedEv のメソッド

フィールド	メソッド	説明
javax.telephony.Address	getAddress()	プロバイダー ドメインから削除されるアドレスと、ユーザ制御リストから削除されるアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrRemovedFromTerminalEv

CiscoAddrRemovedFromTerminalEv イベントは、次の状況で送信されます。

- 管理者が共用回線の含まれるユーザ制御リストから端末を削除した場合、このイベントがアプリケーションに送信されます。
- エクステンション モビリティ（EM; Extension Mobility）ユーザが共用回線の含まれているプロファイルを持つ端末からログアウトすると、このイベントにより、端末の 1 つが既存のアドレスから削除されることが通知されます。
- ユーザ制御リスト内の端末から共用回線が削除された場合。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrRemovedFromTerminalEv extends CiscoProvEv

フィールド

表 6-47　CiscoAddrRemovedFromTerminalEv のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-48　CiscoAddrRemovedFromTerminalEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	端末から削除されたアドレスを返します。
javax.telephony.Terminal	getTerminal()	アドレスから削除された端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrRestrictedEv

アドレスが監視され、Restriction 状態が Restricted に変更された場合、このイベントがアプリケーションに送信されます。

アプリケーションは、アドレスまたは関連付けられた端末が Cisco Unified Communications Manager Administration で制限されている場合、このイベントを受信します。制限された回線では、アドレスが OUT_OF_SERVICE になり、再び有効になるまで IN_SERVICE に戻りません。アドレスが制限されている場合は、addCallObserver および addObserver によって例外がスローされます。

共用回線では、いくつかの回線だけが制限されて残りは制限されなかった場合、例外はスローされませんが、制限された共用回線はイベントを受け取りません。すべての共用回線が制限された場合は、オブザーバを追加すると、アプリケーションは例外のスローを試行します。オブザーバを追加した後にアドレスが制限された場合、アプリケーションは CiscoAddrOutOfServiceEv を受信し、アドレスが有効にされると IN_SERVICE になります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, CiscoRestrictedEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrRestrictedEv extends CiscoRestrictedEv

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoRestrictedEv から
CAUSE_UNKNOWN, CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION, CAUSE_UNSUPPORTED_PROTOCOL, CAUSE_USER_RESTRICTED

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-49　CiscoAddrRestrictedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	Cisco Unified Communications Manager で Restricted 状態に変更されたアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoAddrRestrictedOnTerminalEv

ユーザが制御リストに共用回線を持っており、これらの回線のいずれかが Cisco Unified CM で制限付きとしてマークされている場合、このイベントが送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, CiscoRestrictedEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoAddrRestrictedOnTerminalEv extends CiscoRestrictedEv

フィールド

表 6-50　CiscoAddrRestrictedOnTerminalEv のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoRestrictedEv から
CAUSE_UNKNOWN, CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION, CAUSE_UNSUPPORTED_PROTOCOL, CAUSE_USER_RESTRICTED

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING,

META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-51　CiscoAddrRestricedOnTerminalEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	制限されるアドレスを返します。
javax.telephony.Terminal	getTerminal()	アドレスが制限される端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoCall

CiscoCall インターフェイスは、Cisco Unified Communications Manager に固有の機能を追加することによって、CallControlCall インターフェイスを拡張します。

Cisco Unified Communications Manager では、すべての Call オブジェクトは、グローバル コール ハンドルという共通識別子を共有する一連のコール レッグで構成されています。JTAPI では、Connection オブジェクトがコール レッグを表し、一連の接続を関連付ける Call オブジェクトに、それらのコール レッグに共通するグローバル コール ハンドルが含まれます。

CiscoCall 内のグローバル コール ハンドルは、CallManagerID プロパティと CallID プロパティを使用してアクセスできます。CallManagerID と CallID が組み合わされてグローバル コール ハンドルが形成され、Cisco Unified Communications Manager によって管理されます。このプロパティのペアは、すべての ACTIVE Call オブジェクトを通じて、常に一意になると見なしますが、ACTIVE だったコールが INACTIVE になると、新たに作成された Call オブジェクトを識別するために、その CallManagerID と CallID が再利用される場合があります。したがって、現在 INACTIVE の Call オブジェクトと、現在 ACTIVE の Call オブジェクトが、同じ CallManagerID プロパティと CallID プロパティを持つ場合があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	任意の通話者のドロップ（Drop Any Party）機能に新しいメソッド isConference() が追加されました。

スーパーインターフェイス

javax.telephony.Call, javax.telephony.callcontrol.CallControlCall, CiscoObjectContainer

サブインターフェイス

CiscoConsultCall

宣言

public interface CiscoCall extends javax.telephony.callcontrol.CallControlCall, CiscoObjectContainer

フィールド

表 6-52　CiscoCall のフィールド

インターフェイス	フィールド	説明
Static int	CALLSECURITY_AUTHENTICATED	コールのセキュリティ ステータスが認証されています。
Static int	CALLSECURITY_ENCRYPTED	コールのセキュリティ ステータスが暗号化されています。

表 6-52　CiscoCall のフィールド（続き）

インターフェイス	フィールド	説明
Static int	CALLSECURITY_NOTAUTHENTICATED	コールのセキュリティ ステータスが認証されていません。
Static int	CALLSECURITY_UNKNOWN	コールのセキュリティ ステータスが不明です。
Static int	FEATUREPRIORITY_EMERGENCY	機能プライオリティが緊急状態（emergency）です。
Static int	FEATUREPRIORITY_NORMAL	機能プライオリティが通常（normal）です。
Static int	FEATUREPRIORITY_URGENT	機能プライオリティが緊急（urgent）です。
Static int	PLAYTONE_BOTHLOCALANDREMOTE	このオプションが使用される場合、発信者とモニタリング ターゲット（エージェント）の両方に対してトーンが再生されます。
Static int	PLAYTONE_LOCALONLY	このオプションが使用される場合、モニタリング ターゲット（エージェント）だけに対してトーンが再生されます。
Static int	PLAYTONE_NOLOCAL_OR_REMOTE	このオプションが使用される場合、モニタリング ターゲット（エージェント）または発信者に対してトーンが再生されます。
Static int	PLAYTONE_REMOTEONLY	このオプションが使用される場合、発信者だけに対してトーンが再生されます。
Static int	SILENT_MONITOR	このオプションは、サイレント モニタリングが要求されたことを示します。

継承したフィールド

インターフェイス javax.telephony.Call から
ACTIVE, IDLE, INVALID

メソッド

表 6-53　CiscoCall のメソッド

インターフェイス	メソッド	説明
Void	conference(javax.telephony.Call[]otherCalls)	このインターフェイスは、複数のコールをまとめた結果の、1 つのコールに対して発呼されるすべてのコールの参加者の共有体になります。
java.lang.boolean	isConference()	電話会議のコールである場合は true を返します。そうでない場合は false を返します。

表 6-53　CiscoCall のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony. Connection[]	connect (javax.telephony.Terminal origterm, javax.telephony.Address origaddr java.lang.String.dialedDigits int featurePriority)	このメソッドは Call.connect() をオーバーロードします。新しいパラメータ featurePriority を受け付けます。featurePriority パラメータは、次のいずれかになります。 • CiscoCall.FEATUREPRIORITY_NORMAL • CiscoCall.FEATUREPRIORITY_URGENT • CiscoCall.FEATUREPRIORITY_ EMERGENCY 例外： javax.telephony.ResourceUnavailableException, javax.telephony.PrivilegeViolationException, javax.telephony.InvalidPartyException, javax.telephony.InvalidArgumentException, javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException
boolean	getCalledAddressPI()	getCalledAddressPI に関連付けられた Presentation Indicator（PI; プレゼテーション インジケータ）を返します。
CiscoPartyInfo	getCalledPartyInfo()	コールの着側の PartyInfo を返します。
CiscoCallID	getCallID()	CallID は、同じ CallManagerID を持つすべての ACTIVE コールを通じて、一意の識別子です。
boolean	getCallingAddressPI()	getCallingAddressPI に関連付けられた Presentation Indicator（PI）を返します。
int	getCallSecurityStatus()	このインターフェイスはコールの SecurityStatus を返します。
CiscoConference Chain	getConferenceChain()	このコールがチェーニングされた電話会議の場合、CiscoConferenceChain オブジェクトを返します。
javax.telephony. Address	getCurrentCalledAddress()	コールの現在の着側のアドレスを返します。
boolean	getCurrentCalledAddressPI()	CurrentCalledAddress に関連付けられた Presentation Indicator（PI）を返します。
boolean	getCurrentCalledDisplayNamePI()	getCurredCalledDisplayNamePI に関連付けられた Presentation Indicator（PI）を返します。
java.lang.String	getCurrentCalledPartyDisplayName()	このインターフェイスはコールの着側の表示名を返します。
CiscoPartyInfo	getCurrentCalledPartyInfo()	コールの現在の着側の PartyInfo を返します。

表 6-53　CiscoCall のメソッド（続き）

インターフェイス	メソッド	説明
java.lang.String	getCurrentCalledPartyUnicodeDisplayName()	コールの着側の Unicode 表示名を返します。
int	getCurrentCalledPartyUnicodeDisplayNamelocale()	現在のコールの着側の Unicode 表示名のロケールを返します。
javax.telephony.Address	getCurrentCallingAddress()	コールの現在の着側のアドレスを返します。
boolean	getCurrentCallingAddressPI()	getCurrentCallingAddressPI に関連付けられた Presentation Indicator（PI）を返します。
boolean	getCurrentCallingDisplayNamePI()	getCurrentCalledDisplayNamePI に関連付けられた Presentation Indicator（PI）を返します。
java.lang.String	getCurrentCallingPartyDisplayName()	コールの発側の表示名を返します。
CiscoPartyInfo	getCurrentCallingPartyInfo()	現在のコールの発側の PartyInfo を返します。
java.lang.String	getCurrentCallingPartyUnicodeDisplayName()	コールの発側の Unicode 表示名を返します。
int	getCurrentCallingPartyUnicodeDisplayNamelocale()	現在のコールの着側の Unicode 表示名のロケールを返します。
java.lang.String	getGlobalizedCallingParty()	これは globalizedCallingParty を返します。
CiscoPartyInfo	getLastRedirectedPartyInfo()	コールを最後にリダイレクトした通話者の PartyInfo を返します。
boolean	getLastRedirectingAddressPI()	getLastRedirectingAddressPI に関連付けられた Presentation Indicator（PI）を返します。
CiscoPartyInfo	getLastRedirectingPartyInfo()	推奨されません。getLastRedirectedPartyInfo() を使用してください。
javax.telephony.Address	getModifiedCalledAddress()	このインターフェイスは、着信側トランスフォーメーション パターンまたはその他の方法を使用してコールの着側が変更された場合、変更された着信側アドレスを返します。
javax.telephony.Address	getModifiedCallingAddress()	アプリケーションが selectRoute API またはその他の方法を使用して発信者を変更した場合、このインターフェイスは変更された発信側アドレスを返します。
javax.telephony.Connection[]	startMonitor(javax.telephony.Terminal MonitorInitiatorterminal, javax.telephony.Address MonitorInitiatoraddress, int monitorTargetcallid, java.lang.String monitorTargetDN, java.lang.String monitorTargetTerminalName, int monitorType, int playToneDirection)	アプリケーションがモニタリングのターゲットのコールについての情報を持っている場合、このアプリケーションはこのインターフェイスを使用してコールをモニタリングできます。

表 6-53　CiscoCall のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony. Connection[]	startMonitor(javax.telephony.Terminal MonitorInitiatorterminal, javax.telephony.Address MonitorInitiator address, javax.telephony.TerminalConnection term ConnofMonitorTarget, int monitorType, int PlayToneDirection)	アプリケーションがモニタリング ターゲット（エージェント）のアドレスを監視している場合、このアプリケーションはモニタリング ターゲット（エージェント）の端末接続を使用して、モニタリング要求を開始できます。
javax.telephony. Connection	transfer(java.lang.String address, java.lang.String facCode, java.lang.String cmcCode)	このメソッドは、転送アドレスでこれらのコードがコールをオファーすることを要求される場合に facCode (Forced Authorization Code) と cmcCode (Client Matter Code) も受け付けることを除いて、CallControlCall.transfer(String address) インターフェイスと同じです。

（注）　Cisco Unified JTAPI 実装では、CallControlCall.getCalledAddress() がコールの最初の着側、つまりオリジナルの着側を返します。

継承したメソッド

インターフェイス javax.telephony.callcontrol.CallControlCall から

addParty, conference, consult, consult, drop, getCalledAddress, getCallingAddress, getCallingTerminal, getConferenceController, getConferenceEnable, getLastRedirectedAddress, getTransferController, getTransferEnable, offHook, setConferenceController, setConferenceEnable, setTransferController, setTransferEnable, transfer, transfer

インターフェイス interface javax.telephony.Call から

addObserver, connect, getCallCapabilities, getCapabilities, getConnections, getObservers, getProvider, getState, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から

getObject, setObject

パラメータ

- origterm -
- origaddr -
- dialedDigits -
- featurePriority -

会議コントローラ

会議機能が動作するためには、コントローラの端末上の共通の参加者の TerminalConnection として表される、共通の参加者がすべてのコールに属している必要があります。これらの TerminalConnection は、会議コントローラと呼ばれています。コントローラの端末上のコールの 1 つの TerminalConnection だけが CallControlTerminalConnection.TALKING 状態になるため、セカンダリ コール上の TerminalConnection は CallControlTerminalConnection.HELD 状態にある必要があります。このメソッドの呼び出しの結果、すべての会議コントローラ TerminalConnection が 1 つの TerminalConnection にマージされます。

アプリケーションは、CallControlCalll.setConferenceController() メソッドを呼び出して会議コントローラを設定することによって電話会議が設定されたときに、会議コントローラとして動作する端末を設定できます。CalControlCall.getConferenceController() メソッドは、現在の会議コントローラを返します。ない場合は null を返します。最初に会議コントローラが設定されていない場合、会議機能が起動されると、実装により適切な TerminalConnection が選択されます。

電話コールの引数

このメソッドに引数として渡されたセカンダリ コールの参加者は、このメソッドが起動されたコールにすべて移動されます。つまり、セカンダリ コールの参加者の新しい接続と TerminalConnections が、このコールに対して作成されます。セカンダリ コール上の接続と TerminalConnection はそのコールから削除され、コールは Call.INVALID 状態に移行します。

ほかの共用参加者

指定された会議コントローラ以外に、いくつかのコールの一部であるその他のアドレスと端末がある可能性があります。これらのインスタンスでは、両方のコールで共用される参加者は、1 つにマージされます。つまり、このコールでは接続と TerminalConnection が変更されずそのまま維持されます。セカンダリ コールの対応する接続と TerminalConnection は、そのコールから削除されます。

事前条件

1. tc1 をこの Call の会議コントローラにする
2. connection1 = tc1.getConnection() にする
3. tc2 ～ tcN を otherCalls の会議コントローラにする
4. (this.getProvider()).getState() == Provider.IN_SERVICE
5. this.getState() == Call.ACTIVE
6. tc1.getTerminal() == tc2.getTerminal()...=tcN.getTerminal
7. tc1.getCallControlState() == CallControlTerminalConnection.TALKING/HELD
8. tc2-tcN.getCallControlState() == CallControlTerminalConnection.HELD/TALKING
9. this != otherCalls

事後条件

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.ACTIVE
3. otherCall.getState() == INVALID
4. otherCall からマージされる Connection を c[] にする

5. otherCall からマージされる TerminalConnection を tc[] にする
6. このコールに対して作成される新しい一連の接続を new(c) にする
7. このコールに対して作成される新しい一連の TerminalConnection を new(tc) にする
8. this.getConnections() の new(c) 要素
9. new(c).getCallState() == c.getCallState()
10. (this.getConnections()).getTerminalConnections() の new(tc) 要素
11. new(tc).getCallState() == tc.getCallState()
12. c[i].getCallControlState() == CallControlConnection.DISCONNECTED（すべての i に対して）
13. tc[i].getCallControlState() == CallControlTerminalConnection.DROPPED（すべての i に対して）
14. CallInvalidEv が otherCall に対して配信される
15. CallCtlConnDisconnectedEv/ConnDisconnectedEv がすべての c[i] に対して配信される
16. CallCtlTermConnDroppedEv/TermConnDroppedEv がすべての tc[i] に対して配信される
17. ConnCreatedEv がすべての new(c) に対して配信される
18. TermConnCreatedEv がすべての new(tc) に対して配信される
19. 適切なイベントがすべての new(c) および new(tc) に対して配信される

パラメータ

otherCalls：このコール オブジェクトとマージされるその他のコール。

例外

javax.telephony.InvalidArgumentException：指定されたコール オブジェクトが会議に有効ではありません。

javax.telephony.InvalidStateException：プロバイダーが「イン サービス」でないか、コールが「アクティブ」でないか、会議コントローラが正しい状態でないかのいずれかです。

javax.telephony.MethodNotSupportedException：この実装では、このメソッドをサポートしません。

javax.telephony.PrivilegeViolationException：アプリケーションに、このメソッドを起動する適切な権限がありません。

javax.telephony.ResourceUnavailableException：これは、このメソッドの正常な起動に必要な内部リソースがないことを意味します。

関連項目

ConnCreatedEv, TermConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallInvalidEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

javax.telephony.Connection transfer(java.lang.String address java.lang.String facCode, java.lang.String cmcCode)

connect(Terminal, Address, String, CiscoRTPParams) の例外

javax.telephony.InvalidArgumentException, javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupportedException, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailableExceptionこのメソッドは、転送アドレスでこれらのコードがコールをオファーすることを要求される場合に facCode (Forced Authorization Code) と cmcCode (Client Matter Code) も受け付けることを除いて、CallControlCall.transfer(String address) インターフェイスと同じです。1 つのコードだけが必要な場合、他のコードは null 値にする必要がある場合があります。

ユーザがコードを入力しない場合、または無効なコードを入力した場合、コールが提供されない可能性や、platformException に次のエラー コードが含まれる可能性があります。

CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_INVALID
CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_INVALID

このメソッドのオーバーロードされたバージョンは、このコールの現在のすべての参加者を、転送コントローラ参加者の例外と一緒に、別のアドレスに転送します。転送機能が別のコールを保留し、転送を同時に実行するため、これは、「シングル ステップ転送」とも呼ばれます。このメソッドへのアドレス文字引数が有効であり、完全である必要があります。

転送コントローラ

このバージョンのこのメソッドの転送コントローラは、このコールの転送を実行し、転送の完了後にコールからドロップ オフする参加者を表します。転送コントローラは、CallControlTerminalConnection.TALKING 状態にする必要のある TerminalConnection です。

アプリケーションは、CallControlCall.setTransferController() メソッドを介する転送コントローラとして動作する TerminalConnection を制御できます。CallControlCall.getTransferController() メソッドは、現在の転送コントローラを返します。ない場合は null を返します。最初に転送コントローラが設定されていない場合、転送機能が起動されると、実装により適切な TerminalConnection が選択されます。

転送機能が起動されると、転送コントローラが CallControlTerminalConnection.DROPPED 状態になります。これが接続に関連付けられた唯一の TerminalConnection の場合、その接続も CallControlConnection.DISCONNECTED 状態になります。

新しい接続

このメソッドは、コールが転送された側を示す新しい接続を作成し、返します。コールがプロバイダードメインの外に転送され、記録できなくなった可能性がある場合、この接続が null になることがあります。この接続は少なくとも CallControlConnection.IDLE 状態である必要があります。接続の状態はこのメソッドが返す前に「アイドル状態」の次へと進行している可能性があり、イベントに反映されます。この新しい接続は、コールの通常の宛先側の接続として進行します。接続の一般的なシナリオについて Call.connect() メソッドで説明します。

事前条件

1. tc をこのコールの転送コントローラにする
2. (this.getProvider()).getState() == Provider.IN_SERVICE
3. this.getState() == Call.ACTIVE
4. tc.getCallControlState() == CallControlTerminalConnection.TALKING

事後条件

1. newconnection を作成され、返される接続にする
2. Let connection == tc.getConnection()
3. (this.getProvider()).getState() == Provider.IN_SERVICE
4. this.getState() == Call.ACTIVE
5. tc.getCallControlState() == CallControlTerminalConnection.DROPPED

6. connection.getTerminalConnections().length == 1 の場合、connection.getCallControlState() == CallControlConnection.DISCONNECTED
7. null でない場合、newconnection は this.getConnections() の要素
8. null でない場合、newconnection.getCallControlState() で、少なくとも CallControlConnection.IDLE
9. ConnCreatedEv が newconnection に対して配信される
10. CallCtlTermConnDroppedEv/TermConnDroppedEv が tc に対して配信される
11. 他に TerminalConnections が存在しない場合、CallCtlConnDisconnectedEv/ConnDisconnectedEv が接続に対して配信される

パラメータ

- address：コールが転送される着信先アドレス文字列（dialedDigits）。
- facCode：Force Authorization Code。
- cmcCode：Client Matter Code。

戻り値

宛先に関連付けられた新しい接続、または null。

例外

javax.telephony.InvalidArgumentException：転送を制御するように指定された TerminalConnection が有効でないか、このコールの一部ではありません。

javax.telephony.InvalidStateException：プロバイダーが「イン サービス」でないか、コールが「アクティブ」でないか、転送コントローラが「通話中」でないかのいずれかです。

javax.telephony.InvalidPartyException：着信先アドレスが無効であるか、不完全です。

javax.telephony.MethodNotSupportedException：この実装では、このメソッドをサポートしません。

javax.telephony.PrivilegeViolationException：アプリケーションに、このメソッドを起動する適切な権限がありません。

javax.telephony.ResourceUnavailableException：このメソッドの正常な起動に必要な内部リソースがありません。

関連項目

ConnCreatedEv, ConnDisconnectedEv, TermConnDroppedEv, CallCtlConnDisconnectedEv, CallCtlTermConnDroppedEv

getCurrentCalledAddressPIboolean getCurrentCalledAddressPI()CurrentCalledAddress に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示できます。false が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示しません。

getCurrentCalledDisplayNamePIboolean getCurrentCalledDisplayNamePI()
getCurredCalledDisplayNamePI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこの DisplayName をエンド ユーザに表示できます。false が返された場合、アプリケーションはこの DisplayName をエンド ユーザに表示しません。

getCurrentCallingAddressPIboolean getCurrentCallingAddressPI()getCurrentCallingAddressPI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示できます。false が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示しません。

getCurrentCallingDisplayNamePIboolean getCurrentCallingDisplayNamePI()
getCurrentCalledDisplayNamePI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこの DisplayName をエンド ユーザに表示できます。false が返された場合、アプリケーションはこの DisplayName をエンド ユーザに表示しません。

getLastRedirectingAddressPIboolean getLastRedirectingAddressPI()getLastRedirectingAddressPI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示できます。false が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示しません。

getCalledAddressPIboolean getCalledAddressPI()getCalledAddressPI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示できます。false が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示しません。

getCallingAddressPIboolean getCallingAddressPI()getCallingAddressPI に関連付けられた Presentation Indicator（PI）を返します。true が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示できます。false が返された場合、アプリケーションはこのアドレス名をエンド ユーザに表示しません。

getCurrentCalledPartyUnicodeDisplayNamejava.lang.String
getCurrentCalledPartyUnicodeDisplayName()現在のコールの着側の Unicode 表示名を返します。表示名が不明な場合は null を返します。

getCurrentCalledPartyUnicodeDisplayNamelocaleint
getCurrentCalledPartyUnicodeDisplayNamelocale()現在のコールの着側の Unicode 表示名のロケールを返します。CiscoLocale インターフェイスは、サポートされているロケールを列挙します。

getCurrentCallingPartyUnicodeDisplayNamejava.lang.String
getCurrentCallingPartyUnicodeDisplayName()コールの発側の Unicode 表示名を返します。表示名が不明な場合は null を返します。

getCurrentCallingPartyUnicodeDisplayNamelocaleint
getCurrentCallingPartyUnicodeDisplayNamelocale()現在のコールの着側の Unicode 表示名のロケールを返します。

getCurrentCallingPartyInfoCiscoPartyInfo getCurrentCallingPartyInfo()現在のコールの発側の PartyInfo を返します。

getCurrentCalledPartyInfoCiscoPartyInfo getCurrentCalledPartyInfo()コールの現在の着側の PartyInfo を返します。

getLastRedirectingPartyInfoCiscoPartyInfo getLastRedirectingPartyInfo()推奨されません。getLastRedirectedPartyInfo() を使用してください。

コールを最後にリダイレクトした通話者の PartyInfo を返します。

getLastRedirectedPartyInfoCiscoPartyInfo getLastRedirectedPartyInfo()コールを最後にリダイレクトした通話者の PartyInfo を返します。

getCalledPartyInfoCiscoPartyInfo getCalledPartyInfo()コールの着側の PartyInfo を返します。

javax.telephony.Connection[]startMonitor(javax.telephony.Terminal MonitorInitiatorterminal,
javax.telephony.Address MonitorInitiatoraddress,
javax.telephony.TerminalConnection termConnofMonitorTarget, int monitorType,
int PlayToneDirection)

throws

javax.telephony.ResourceUnavailableException, javax.telephony.PrivilegeViolationException,
javax.telephony.InvalidPartyException, javax.telephony.InvalidArgumentException,
javax.telephony.InvalidStateException, javax.telephony.MethodNotSupportedException

アプリケーションがモニタリング ターゲット（エージェント）のアドレスを監視している場合、このアプリケーションはモニタリング ターゲット（エージェント）の端末接続を使用して、モニタリング要求を開始できます。このインターフェイスは発側のエンドポイントからコールを発信し、モニタリング ターゲットのコールをモニタリングします。

事前条件

1. (this.getProvider()).getState() == Provider.IN_SERVICE
2. this.getState() == Call.IDLE
3. ((CiscoProviderCapabilities)(this.getTerminal().getProvider().getProviderCapabilities()).canMonitor( ) == TRUE
4. TerminalConnection.getProvider() == this.getProvider()

パラメータ

- MonitorInitiatorterminal：発呼側の端末。
- MonitorInitiatoraddress：発呼側の端末アドレス。
- termConnofMonitorTarget：ターゲットの TerminalConnection。
- monitorType：モニタリングのタイプ。CiscoCall.SILENT_MONITOR を使用します。
- PlayToneDirection：トーンをターゲット、発信側、または両方で再生するかを示します。使用可能な値は、CiscoCall.PLAYTONE_NOLOCAL_OR_REMOTE、CiscoCall.PLAYTONE_LOCALONLY、CiscoCall.PLAYTONE_REMOTEONLY、または CiscoCall.PLAYTONE_BOTHLOCALANDREMOTE です。

例外

javax.telephony.ResourceUnavailableException

javax.telephony.PrivilegeViolationException

javax.telephony.InvalidPartyException

javax.telephony.InvalidArgumentException

javax.telephony.InvalidStateException

javax.telephony.MethodNotSupportedException

関連資料

詳細については、CallControlCall を参照してください。

CiscoCallChangedEv

CiscoCallChangedEv イベントは、すべてのサポートされる機能でコールの Global Call ID（GCID）が変更されるたびに、コール オブザーバに配信されます。CiscoCallChangedEv は、パス交換（(QSIG_PR）およびその他の機能（転送、会議、割り込み、パーク解除など）のためにコールの GCID が変更された場合に配信されます。共用回線の場合、複数の CiscoCallChangedEv イベントが配信されます。

このイベントは、2 つ以上のコールを 1 つのコールにマージしたときに配信されます。転送、会議、パーク解除、割り込み、および C 割り込みによってこのイベントが開始されます。アプリケーションは CiscoCallEv.getCiscoFeatureReason() を起動して、このイベントが発生した原因である機能コードを検出できます。

このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoCallChangedEv extends CiscoCallEv

フィールド

表 6-54　CiscoCallChangedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,

CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,
CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL,
CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS,
CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER,
CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR,
CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING,
CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION,
CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING,
CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL,
CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER,
CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR,
CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR,
CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL,
CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED,
CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY,
CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED,
CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT,
CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION,
CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-55　CiscoCallChangedEv のメソッド

インターフェイス	メソッド	説明
CiscoConnection	getConnection()	変更が発生したアドレスに CiscoConnection を返します。
CiscoCall	getOriginalCall()	INVALID 状態にするコールを返します。
CiscoCall	getSurvivingCall()	callID の変更後も有効のままになっているコールを返します。
javax.telephony. TerminalConnection	getTerminalConnection()	変更が発生した TerminalConnection を返します。この値は、アドレスに対して TerminalConnection が作成される前にコール ID が変更された場合、null である可能性があります。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoCallConsultCancelledEv

このイベントはアプリケーションに、操作のキャンセルが発生したことを通知します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	スワップ/キャンセルの新しいイベント：転送/会議動作の変更機能。

スーパーインターフェイス

なし

宣言

public interface CiscoCallConsultCancelledEv

フィールド

なし

継承したフィールド

なし

メソッド

表 6-56　CiscoCallConsultCancelledEv のメソッド

インターフェイス	メソッド	説明
CiscoCall	getConsultCall()	コンサルト オペレーションがキャンセルされるコンサルト コールを返します。コンサルト コールが存在しない場合、NULL を返します。 このコール イベントの getCall() API は親コールを返します。

継承したメソッド

なし

関連資料

なし。

CiscoCallCtlConnOfferedEv

CiscoCallCtlConnOfferedEv インターフェイスは CallCtlConnOfferedEv インターフェイスを拡張して、アプリケーションが発信側端末の IP アドレスを取得できるようにします。すべての発信側デバイスの IP アドレス情報が参照可能とは限りません。戻り値が 0（または null）の場合、情報が取得可能でないことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlConnEv, javax.telephony.callcontrol.events.CallCtlConnOfferedEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.events.CallEv, javax.telephony.events.ConnEv, javax.telephony.events.Ev

宣言

public interface CiscoCallCtlConnOfferedEv extends javax.telephony.callcontrol.events.CallCtlConnOfferedEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.callcontrol.events.CallCtlConnOfferedEv から
なし

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から
CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-57　CiscoCallCtlConnOfferedEv のメソッド

インターフェイス	メソッド	説明
java.net.InetAddress	getCallingPartyIpAddr()	発信側の IP アドレスか、IP アドレスが取得できない場合は 0（または null）を返します。

継承したメソッド

インターフェイス javax.telephony.callcontrol.events.CallCtlCallEv から
getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から
getCallControlCause

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ConnEv から
getConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

なし

CiscoCallCtlTermConnHeldReversionEv

CiscoCallCtlTermConnHeldReversionEv イベントは、Cisco Unified Communications Manager から TerminalConnection で保留復帰通知を受信したことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcontrol.events.CallCtlCallEv, javax.telephony.callcontrol.events.CallCtlEv, javax.telephony.callcontrol.events.CallCtlTermConnEv, javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoCallCtlTermConnHeldReversionEv extends javax.telephony.callcontrol.events.CallCtlTermConnEv

フィールド

表 6-58　CiscoCallCtlTermConnHeldReversionEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から

CAUSE_ALTERNATE, CAUSE_BUSY, CAUSE_CALL_BACK, CAUSE_CALL_NOT_ANSWERED, CAUSE_CALL_PICKUP, CAUSE_CONFERENCE, CAUSE_DO_NOT_DISTURB, CAUSE_PARK, CAUSE_REDIRECTED, CAUSE_REORDER_TONE, CAUSE_TRANSFER, CAUSE_TRUNKS_BUSY, CAUSE_UNHOLD

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE,

CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.callcontrol.events.CallCtlCallEv から
getCalledAddress, getCallingAddress, getCallingTerminal, getLastRedirectedAddress

インターフェイス javax.telephony.callcontrol.events.CallCtlEv から
getCallControlCause

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoCallEv

JTAPI のコアである javax.telephony.events.CallEv インターフェイスを拡張する CiscoCallEv インターフェイスは、Cisco によって拡張されたすべての JTAPI Call イベントの基本インターフェイスになります。このパッケージのコール関連イベントはすべて、直接的または間接的にこのインターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoEv, javax.telephony.events.Ev

サブインターフェイス

CiscoCallChangedEv, CiscoCallSecurityStatusChangedEv, CiscoConferenceChainAddedEv, CiscoConferenceChainRemovedEv, CiscoConferenceEndEv, CiscoConferenceStartEv, CiscoConsultCallActiveEv, CiscoToneChangedEv, CiscoTransferEndEv, CiscoTransferStartEv

宣言

public interface CiscoCallEv extends CiscoEv, javax.telephony.events.CallEv

フィールド

表 6-59　CiscoCallEv のフィールド

インターフェイス	フィールド	説明
Static int	CAUSE_ACCESSINFORMATION DISCARDED	これは、リモート ユーザが要求したときに、ネットワークがアクセス情報を配信できないことを示します。
Static int	CAUSE_BARGE	コールが割り込みコールであることを示します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
staticint	CAUSE_BCBPRESENTLYAVAIL	この原因は、ユーザがベアラ機能を要求したことを示します。ベアラ機能は、この原因を生成した装置によって実装されますが、この時点では利用できません。
static int	CAUSE_BCNAUTHORIZED	この原因は、ユーザがベアラ機能を要求したことを示します。ベアラ機能は、この原因を生成した装置によって実装されますが、このユーザは使用を許可されていません。
static int	CAUSE_BEARERCAPNIMPL	この原因は、この原因を送信した装置が、要求されたベアラ機能をサポートしていないことを示します。
static int	CAUSE_CALLBEINGDELIVERED	この原因は、ユーザに対して着信コールがあり、その着信コールが、同様のコールのためにそのユーザに対してすでに確立されているチャネルに接続されることを示します。
static int	CAUSE_CALLIDINUSE	この原因は、ネットワークが、コールが再開される可能性があるインターフェイスのドメイン内にある一時停止されたコールですでに使用されているコールの ID（null コール ID を含む）を含むコールの一時停止要求を受け取ったことを意味します。
static int	CAUSE_CALLMANAGER_FAILURE	この原因は、コール マネージャの障害によって発生した障害を示します。
static int	CAUSE_CALLREJECTED	この原因は、この原因を送信した装置が、このコールの受け入れを拒否していることを示します。
static int	CAUSE_CALLSPLIT	この原因は、コール スプリットを示します。つまり、会議または転送です。
static int	CAUSE_CHANTYPENIMPL	この原因は、この原因を送信した装置が、要求されたチャネル タイプをサポートしていないことを示します。
static int	CAUSE_CHANUNACCEPTABLE	この原因は、最後に指定されたチャネルでは、このコールで使用するための送信エンティティが受け入れられないことを示します。
static int	CAUSE_CTICCMSIP400BADREQUEST	この原因は、不正な要求のためにコールが拒否されることを示します。
static int	CAUSE_CTICCMSIP401UNAUTHORIZED	この原因は、要求は有効であるものの、権限を持っていないことを示します。
static int	CAUSE_CTICCMSIP402PAYMENT REQUIRED	これは、使用するためには支払いが必要であることを示します。
static int	CAUSE_CTICCMSIP403FORBIDDEN	この原因は、サーバが要求を認識したものの、対応を拒否していることを示します。
static int	CAUSE_CTICCMSIP404NOTFOUND	この原因は、サーバで要求の URI を検出できなかったことを示します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_CTICCMSIP405METHODNOTALLOWED	この原因は、Request-Line で指定されたメソッドが認識されたものの、Request-URI で指定されたアドレスで許可されていないことを示します。
static int	CAUSE_CTICCMSIP406NOTACCEPTABLE	この原因は、要求の要件が満たされていないために、要求を処理できないことを意味します。
static int	CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED	この原因は、要求を行うための権限がなく、要求のためにはプロキシ認証が必要であることを示します。
static int	CAUSE_CTICCMSIP408REQUESTTIMEOUT	この原因は、要求に対するタイム アウト エラーを意味します。
static int	CAUSE_CTICCMSIP410GONE	この原因は、要求されたリソースがサーバで利用できなくなっており、転送アドレスが不明であることを示します。
static int	CAUSE_CTICCMSIP411LENGTHREQUIRED	この原因は、インターワーキング メッセージの長さが必要であることを示します。
static int	CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG	この原因は、要求のエンティティ自体が、サーバが処理を想定しているサイズ、または処理できるサイズよりも大きいために、サーバが要求の処理を拒否していることを示します。
static int	CAUSE_CTICCMSIP414REQUESTURITOOLONG	この原因は、Request-URI がサーバが解釈を想定している長さ、または解釈できる長さよりも長いために、サーバが要求の処理を拒否していることを示します。
static int	CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE	この原因は、要求のメッセージの本文で、要求されたメソッドのために、サーバでサポートされないメディア タイプが記述されているために、サーバが要求の処理を拒否していることを意味します。
static int	CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME	この原因は、Request-URI の URI のスキームがサーバで認識されないために、サーバが要求を処理できないことを示します。
static int	CAUSE_CTICCMSIP420BADEXTENSION	この原因は、Proxy-Require または Require ヘッダー フィールドで指定されたプロトコル拡張がサーバで認識されないことを示します。
static int	CAUSE_CTICCMSIP421EXTENSTIONREQUIRED	この原因は、UAS が要求を処理するために特別な拡張が必要であるものの、この拡張が要求の Supported ヘッダー フィールドの一覧に表示されていないことを示します。
static int	CAUSE_CTICCMSIP423INTERVALTOOBRIEF	この原因は、要求によってリフレッシュされるリソースの期限切れ時間が短すぎるために、サーバが要求を拒否していることを示します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_CTICCMSIP480TEMPORARILY UNAVAILABLE	この原因は、着信側のエンド システムが正常に通信できるものの、着信側が現在、利用不能である（たとえば、ログインしていない、ログインしているが、着信側との通信が不可能な状態にある、または「Do Not Disturb（サイレント)」機能が有効になっている）ことを示します。
static int	CAUSE_CTICCMSIP481CALLLEGDOES NOTEXIST	この原因は、UAS が既存のダイアログまたはトランザクションと一致していない要求を受け取ったことを意味します。
static int	CAUSE_CTICCMSIP482LOOPDETECTED	この原因は、サーバがループを検出したことを示します。
static int	CAUSE_CTICCMSIP483TOOMANYHOOPS	この原因は、サーバが、Max-Forwards ヘッダー フィールドの値がゼロ（または実際のホップよりも小さい値）である要求を受け取ったことを示します。
static int	CAUSE_CTICCMSIP484ADDRESSIN COMPLETE	この原因は、サーバが Request-URI が不完全な要求を受け取ったことを示します。
static int	CAUSE_CTICCMSIP485AMBIGUOUS	この原因は、Request-URI があいまいであることを示します。
static int	CAUSE_CTICCMSIP486BUSYHERE	この原因は、着信側のエンド システムが正常に通信できるものの、着信側が現在、このエンド システムで追加のコールを受け入れることを拒否しているか、受け入れることができないことを示します。
static int	CAUSE_CTICCMSIP487REQUEST TERMINATED	この原因は、この要求が BYE 要求または CANCEL 要求によって終了したことを示します。
static int	CAUSE_CTICCMSIP488NOTACCEPTABLE HERE	この原因は、606（受け入れられない）と同じ意味であるものの、Request-URI によって指定された特定のリソースだけに適用され、要求が成功する場合もあることを示します。
static int	CAUSE_CTICCMSIP491REQUEST PENDING	この原因は、同じダイアログ内で保留中の要求がある UAS によって要求が受け入れられたことを示します。
static int	CAUSE_CTICCMSIP493 UNDECIPHERABLE	この原因は、受信者が適切な復号化キーを保持していないか、提供されていない、暗号化された MIME 本文を含む UAS によって要求が受け入れられたことを示します。
static int	CAUSE_CTICCMSIP500SERVERINTERNAL ERROR	この原因は、サーバで要求の処理を妨げる予期しない条件が発生したことを示します。
static int	CAUSE_CTICCMSIP501NOT IMPLEMENTED	この原因は、サーバが要求を処理するために必要な機能をサポートしていないことを示します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_CTICCMSIP502BADGATEWAY	この原因は、ゲートウェイまたはプロキシの役割を果たしているサーバが、要求の処理を試行するためにアクセスしたダウンストリーム サーバから無効な応答を受信したことを示します。
static int	CAUSE_CTICCMSIP503SERVICEUNAVAILABLE	この原因は、一時的なサーバのオーバーロードまたはメンテナンスのために、サーバが一時的に要求を処理できないことを示します。
static int	CAUSE_CTICCMSIP504SERVERTIMEOUT	この原因は、サーバが要求の処理を試行するためにアクセスした外部サーバから応答をタイムリーに受信できないことを示します。
static int	CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED	この原因は、サーバが、要求で使用された SIP プロトコル バージョンをサポートしていないか、サポートを拒否していることを示します。
static int	CAUSE_CTICCMSIP513MESSAGETOOLARGE	この原因は、メッセージの長さがサーバの機能を超えたために、サーバが要求を処理できないことを示します。
static int	CAUSE_CTICCMSIP600BUSYEVERYWHERE	この原因は、着信側のエンド システムが正常に通信しているものの、着信側がビジーであり、この時点でコールに応答しようとしていないことを示します。
static int	CAUSE_CTICCMSIP603DECLINE	この原因は、着信側のマシンが正常に通信しているものの、ユーザが明示的に参加しようとしていないか、参加できないことを示します。
static int	CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE	この原因は、ユーザが Request-URI でどこにも存在していないと示した認証情報をサーバが持っていることを示します。
static int	CAUSE_CTICCMSIP606NOTACCEPTABLE	この原因は、ユーザのエージェントが正常に通信しているものの、セッションに関する要求されたメディア、帯域幅、またはアドレス指定形式などが受け入れられなかったことを示します。
static int	CAUSE_CTICONFERENCEFULL	この原因は、電話会議がいっぱいで、参加者を追加できないことを示します。
static int	CAUSE_CTIDEVICENOTPREEMPTABLE	この原因は、デバイスのプリエンプション処理ができないことを示します。
static int	CAUSE_CTIDROPCONFEREE	この原因は、参加者が会議からドロップしたことによる接続解除を示します。
static int	CAUSE_CTIMANAGER_FAILURE	この原因は、CTI マネージャの障害によって発生した障害を意味します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_CTIPRECEDENCECALLBLOCKED	この原因は、予測可能な回路がないか、着信側が回避可能レベルが同じか、それより高いコールによってビジーであることを示します。
static int	CAUSE_CTIPRECEDENCELEVELEXCEEDED	この原因は、コールの優先順位レベルが認証レベルを超えていることを示します。
static int	CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH	この原因は、優先コールが帯域幅不足で先に進めることができないことを示します。
static int	CAUSE_CTIPREEMPTFORREUSE	この原因は、コールが優先され、回路が優先交換で再利用されるために予約されていることを示します。
static int	CAUSE_CTIPREEMPTNOREUSE	この原因は、コールが優先されることを示します。
static int	CAUSE_DESTINATIONOUTOFORDER	この原因は、ユーザが指定した宛先に、その宛先へのインターフェイスが正常に機能していないためにアクセスできないことを示します。
static int	CAUSE_DESTNUMMISSANDDCNOTSUB	この原因は、指定された CUG が存在しないことを示します。
static int	CAUSE_DPARK	コールがダイレクト パークされたコールであることを示します。
static int	CAUSE_DPARK_REMINDER	ダイレクト パークされたコールのリマインダ コールであることを示します。
static int	CAUSE_DPARK_UNPARK	ダイレクト パークされたコールが、現在パーク解除されていることを示します。
static int	CAUSE_EXCHANGEROUTINGERROR	この原因は、交換機が、指定された宛先にコールをルーティングできなかったことを示します。
static int	CAUSE_FAC_CMC	Forced Authorization Code（FAC）または Client Matter Code（CMC）がコールのルーティングに必要であることを示します。
static int	CAUSE_FACILITYREJECTED	この原因は、ユーザが要求した補足サービスがネットワークによって提供されない場合に返されます。
static int	CAUSE_IDENTIFIEDCHANDOESNOTEXIST	この原因は、この原因を送信した装置が、コール用のインターフェイス上でアクティブになっていないチャネルを使用するという要求を受け取ったことを示します。
static int	CAUSE_IENIMPL	この原因は、この原因を送信した装置が、認識されない情報要素/パラメータが含まれるメッセージを受信したことを示します。これは、情報要素/パラメータが定義されていないか、または定義されているものの、この原因を送信した装置によって実装されないためです。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_INBOUNDBLINDTRANSFER	コールが着信ブラインド転送コールであることを示します。
static int	CAUSE_INBOUNDCONFERENCE	コールが着信会議コールであることを示します。
static int	CAUSE_INBOUNDTRANSFER	コールが着信転送コールであることを示します。
static int	CAUSE_INCOMINGCALLBARRED	この原因は、この番号への着信コールが拒否されたことを示します。
static int	CAUSE_INCOMPATABLEDDESTINATION	この原因は、この原因を送信した装置が、低いレイヤとの互換性を持つコールを確立するという要求を受け取ったことを示します。
static int	CAUSE_INTERWORKINGUNSPECIFIED	この原因は、インターワーキング コールが終了したことを示します。
static int	CAUSE_INVALIDCALLREFVALUE	この原因は、この原因を送信した装置が、現在ユーザネットワーク インターフェイスで使用されていないコール参照を持つメッセージを受信したことを示します。
static int	CAUSE_INVALIDIECONTENTS	この原因は、この原因を送信した装置が、実装したものの、1 つ以上のフィールドがこの原因を送信した装置によって実装されなかった方法でコード化される情報要素を受け取ったことを示します。
static int	CAUSE_INVALIDMESSAGEUNSPECIFIED	この原因は、無効なメッセージ クラスの他の原因が適用されない場合にだけ、無効なメッセージ イベントを報告するために使用されます。
static int	CAUSE_INVALIDNUMBERFORMAT	この原因は、着信側の番号が有効な形式ではないか、不完全であるために、この着信側にアクセスできないことを示します。
static int	CAUSE_INVALIDTRANSITNETSEL	この原因は、不正な形式の中継ネットワーク ID を受信しました。
static int	CAUSE_MANDATORYIEMISSING	この原因は、この原因を送信した装置が、情報要素が欠落しているメッセージを受信し、欠落している部分があると、このメッセージを処理できないことを示します。
static int	CAUSE_MSGNCOMPATABLEWCS	この原因は、このコールの状態との互換性がないメッセージを受信したことを示します。
static int	CAUSE_MSGTYPENCOMPATWCS	この原因は、この原因を送信した装置が、手順でこのコールの状態で受信できるメッセージであることが示されていないメッセージを受信したか、または互換性のないコールの状態を示す STATUS メッセージを受信したことを示します。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_MSGTYPENIMPL	この原因は、この原因を送信した装置が、定義されていないメッセージであるか、定義されているものの、この原因を送信した装置で実装されていないために認識されないメッセージ タイプのメッセージを受信したことを示します。
static int	CAUSE_NETOUTOFORDER	この原因は、ネットワークが正常に動作していないことと、この状態が比較的長い時間続くことが予測されるということを示します。
static int	CAUSE_NOANSWERFROMUSER	この原因は、着信側に警告が発せられているものの、所定の期間内の応答指示に応じない場合に使用されます。
static int	CAUSE_NOCALLSUSPENDED	この原因は、ネットワークが、現在インターフェイスのドメイン内で一時停止されている（復帰可能な）コールの存在することを示していないコール ID 情報要素を含むコール復帰要求を受け取ったことを示します。
static int	CAUSE_NOCIRCAVAIL	この原因は、現在、コールを処理するために利用できる適切な回路/チャネルがないことを示します。
static int	CAUSE_NOERROR	これは通常、エラーが発生しておらず、オペレーションが正常に完了する場合に指定されます。
static int	CAUSE_NONSELECTEDUSERCLEARING	この原因は、ユーザ対して着信コールがなかったことを示します。
static int	CAUSE_NORMALCALLCLEARING	この原因は、コールに関与しているいずれかのユーザがコールのクリアを要求したために、コールがクリアされたことを示します。
static int	CAUSE_NORMALUNSPECIFIED	この原因は、通常のクラスのその他の原因が適用されない場合にだけ、通常のイベントを報告するために使用されます。
static int	CAUSE_NOROUTETODDESTINATION	この原因は、コールがルーティングされたネットワークが、目的の宛先に対してサービスを提供していないために、この着信側にアクセスできないことを示します。
static int	CAUSE_NOROUTETOTRANSITNET	この原因は、この原因を送信した装置が、認識していない特定の中継ネットワークを介したコールのルーティング要求を受け取ったことを示します。
static int	CAUSE_NOUSERRESPONDING	この原因は、着信側が、割り当てられた期間内にアラートまたは接続が指示されたコール確立メッセージに応答しない場合に使用されます。
static int	CAUSE_NUMBERCHANGED	この原因は、発信側によって指定された着信側番号が、現在では割り当てられていない場合に、着信側に返されます。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_ONLYRDIVEARERCAPAVAIL	この原因は、発信側が制限解除ベアラ サービスを要求したものの、この原因を送信した装置が、要求されたベアラ機能の制限バージョンだけをサポートしていることを示します。
static int	CAUSE_OUTBOUNDCONFERENCE	コールが発信会議コールであることを示します。
static int	CAUSE_OUTBOUNDTRANSFER	コールが発信転送コールであることを示します。
static int	CAUSE_OUTOFBANDWIDTH	この原因は、帯域幅不足のためにコールを処理できなかったことを示します。
static int	CAUSE_PROTOCOLERRORUNSPECIFIED	この原因は、プロトコル エラー クラスのその他の原因が適用されない場合にだけ、プロトコル エラーを報告するために使用されます。
static int	CAUSE_QSIG_PR	これは、コールの QSIG パス置換を示します。
static int	CAUSE_QUALOFSERVNAVAIL	この原因は、推奨事項 X.213 で定義されている、要求された Quality of Service（QoS）を報告するために使用されます。
static int	CAUSE_QUIET_CLEAR	これは、コール マネージャがダウンしたためにコールがクリアされたものの、エンドポイント間のメディアは接続されたままであることを示します。
static int	CAUSE_RECOVERYONTIMEREXPIRY	この原因は、エラー処理手順に関連付けられたタイマーの満了によって手順が開始されたことを示します。
static int	CAUSE_REDIRECTED	この原因は、コールが別の人にリダイレクトされることを示します。
static int	CAUSE_REQCALLIDHASBEENCLEARED	この原因は、ネットワークが、一時停止されていたコールが（ネットワークのタイムアウトまたはリモート ユーザによって）一時停止中にクリアされたことを示すコール ID 情報要素を含むコール復帰要求を受け取ったことを示します。
static int	CAUSE_REQCIRCNAVIL	この原因は、要求エンティティによって指定された回路またはチャネルが、インターフェイスの他の側で提供できない場合に返されます。
static int	CAUSE_REQFACILITYNIMPL	この原因は、この原因を送信した装置が、要求された内容をサポートしていないことを示します。
static int	CAUSE_REQFACILITYNOTSUBSCRIBED	この原因は、ユーザが補足サービスを要求したことを示します。補足サービスは、この原因を生成した装置によって実装されますが、このユーザは使用を許可されていません。

表 6-59　CiscoCallEv のフィールド （続き）

インターフェイス	フィールド	説明
static int	CAUSE_RESOURCESNAVAIL	この原因は、リソースを利用できないというイベントを報告するために使用されます。
static int	CAUSE_RESPONSETOSTATUSENQUIRY	この原因は、STATUS メッセージが生成された理由が STATUS INQUIRY の受信の前である場合、STATUS メッセージに含まれます。
static int	CAUSE_SERVNOTAVAILUNSPECIFIED	この原因は、利用できないクラスのサービスまたはオプションのその他の原因が適用されない場合にだけ、サービスまたはオプションを利用できないというイベントを報告するために使用されます。
static int	CAUSE_SERVOPERATIONVIOLATED	この原因は、発信側が発信 CUG コールの CUG のメンバであっても示します。
static int	CAUSE_SERVOROPTNAVAILORIMPL	この原因は、実装されないクラスのサービスまたはオプションのその他の原因が適用されない場合にだけ、サービスまたはオプションが実装されないというイベントを報告するために使用されます。
static int	CAUSE_SUBSCRIBERABSENT	この原因の値は、モバイル端末がログオフした場合に使用されます。
static int	CAUSE_SUSPCALLBUTNOTTHISONE	この原因は、現在、一時停止中のコールで使用されているコール ID とは異なるコール ID でコールの復帰が試行されたことを示します。
static int	CAUSE_SWITCHINGEQUIPMENT CONGESTION	この原因は、この原因を生成した装置の切り替えによってトラフィックが増大することを示します。
static int	CAUSE_TEMPORARYFAILURE	この原因は、ネットワークが正常に動作していないことと、この状態が長い時間続き、ユーザがすぐにでも別のコールを試行することが予測されるということを示します。
static int	CAUSE_UNALLOCATEDNUMBER	この原因は、発信側のユーザによって要求された宛先が、無効な番号であるためにアクセスできないことを示します。
static int	CAUSE_USERBUSY	この原因は、着信側のユーザがビジー状態であるために、別のコールを受け入れることができないことを示すために使用されます。

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE,

CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

メソッド

表 6-60　CiscoCallEv のメソッド

インターフェイス	メソッド	説明
Int	getCiscoCause()	このイベントに関する Cisco Unified Communications Manager の原因を返します。監視対象のエンドポイントでイベントが発生した原因を識別しないと、適切に機能しないアプリケーションもあります。たとえば、コールに応答がなかったために接続が解除されたのか（CAUSE_NOANSWERFROMUSER）、あるいはコールが拒否されたために接続が解除されたのか（CAUSE_CALLREJECTED）を識別しなければならない場合があります。戻り値：このイベントに関する Cisco Unified Communications Manager の原因

表 6-60　CiscoCallEv のメソッド

インターフェイス	メソッド	説明
Int	getCiscoFeatureReason()	このイベントに関する Cisco Unified Communications Manager の機能原因を返します。イベントが発生した原因を識別しないと、適切に機能しないアプリケーションもあります。このインターフェイスは、現行の機能および新機能における JTAPI Call イベントに CiscoFeatureReason を提供します。転送などの既存の機能は、現状どおり以前の CiscoCallEv.getCiscoCause() インターフェイスから CiscoCause を受け取ります。このインターフェイスは転送の REASON_TRANSFER を提供します。注意：アプリケーションは、未知の原因を処理してデフォルト動作を提示できる必要があります。これは将来新しい原因が追加され、このインターフェイスの下位互換性がなくなる可能性があるためです。指定可能な値は CiscoFeatureReason インターフェイスで定義されています。戻り値：このイベントに関する Cisco Unified Communications Manager の機能原因

関連資料

詳細については、「定数フィールド値」（P.F-1）と CallEv を参照してください。

CiscoCallFeatureCancelledEv

このイベントはアプリケーションに、操作のキャンセルが発生したことを通知します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoCallFeatureCancelledEv

メソッド

表 6-61　CiscoCallFeatureCancelledEv のメソッド

インターフェイス	メソッド	説明
CiscoCall	getConsultCall()	コンサルト オペレーションがキャンセルされるコンサルト コールを返します。コンサルト コールが存在しない場合、NULL を返します。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoCallID

CiscoCallID オブジェクトは、各コールに関連付けられる一意のオブジェクトです。アプリケーションでは、オブジェクト自体または intValue() メソッドで返されたオブジェクトの整数表現を使用できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoObjectContainer

宣言

Public interface CiscoCallID extends CiscoObjectContainer

フィールド

なし

メソッド

表 6-62　CiscoCallID のメソッド

インターフェイス	メソッド	説明
Int	intValue()	このオブジェクトの整数表現を返します。戻り値：このオブジェクトの Int An 整数表現
CiscoCall	getCall()	この CiscoCallID に対応している CiscoCall を返します。
int	getCallManagerID()	この CiscoCallID に関連付けられたコールの Cisco Unified Communications Manager NodeID を返します。
int	getGlobalCallID()	この CiscoCallID に関連付けられたコールの GlobalCallID を返します。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

なし

CiscoMediaCallSecurityIndicator

CiscoMediaCallSecurityIndicator では、コールのセキュリティ インジケータを取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoMediaCallSecurityIndicator

フィールド

なし

メソッド

表 6-63　CiscoMediaCallSecurityIndicator のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	CiscoCallID を返します。
int	getCiscoMediaSecurityIndicator()	次の定数の 1 つのメディアのセキュリティ インジケータを返します。 CiscoMediaSecurityIndicator.MEDIA_ENCRYPT_USER_NOT_AUTHORIZED CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_UNAVAILABLE CiscoMediaSecurityIndicator.MEDIA_NOT_ENCRYPTED
CiscoRTPHandle	getCiscoRTPHandle()	CiscoProvider.getCall を使用して CiscoRTPHandle object.Applications が取得できるコール参照を返します。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。

関連資料

CiscoRTPParams を参照してください。

CiscoCallSecurityStatusChangedEv

コール全体のセキュリティ ステータスが変更されると、アプリケーションは CiscoCallSecurityStatusChangedEv を受信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoCallSecurityStatusChangedEv extends CiscoCallEv

フィールド

表 6-64　CiscoCallSecurityStatusChangedEv のフィールド

インターフェイス	フィールド
Static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,

CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-65　CiscoCallSecurityStatusChangedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.events.Ev	getID()	指定元：interface javax.telephony.events.Ev の getID
getCallSecurityStatus()	getCallSecurityStatus()	コール セキュリティ ステータスを返します。このインターフェイスは、以下を返すことができます。CiscoCall.CALLSECURITY_UNKNOWN, CiscoCall.CALLSECURITY_NOTAUTHENTICATED, CiscoCall.CALLSECURITY_AUTHENTICATED, CiscoCall.CALLSECURITY_ENCRYPTED

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoConferenceChain

このインターフェイスは、会議チェーンで繋がれている電話会議の会議チェーン接続へのリンクを提供します。このオブジェクトは CiscoConferenceChainAddedEv および CiscoConferenceChainRemovedEv から取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoConferenceChain

フィールド

なし

メソッド

表 6-66　CiscoConferenceChain のメソッド

インターフェイス	メソッド	説明
javax.telephony.Connection[]	getChainedConferenceConnections()	1 つの会議にチェーニングされている電話会議の接続の配列を返します。アプリケーションは、このリストを使用してすべての繋がれている電話会議を取得できます。会議にチェーニングされたすべての接続のリストを取得するには、プロバイダーは、各会議で最低 1 つの通話者にオブザーバを必要とします。
CiscoCall[]	getChainedConferenceCalls()	1 つの会議にチェーニングされているコールの配列を返します。このインターフェイスでは、会議チェーンに含まれているプロバイダーで監視されているコールだけが返されます。

関連資料

詳細については、CiscoConferenceChainAddedEv と CiscoConferenceChainRemovedEv を参照してください。

CiscoConferenceChainAddedEv

会議チェーン接続がコールに追加されると、CiscoConferenceChainAddedEv イベントが送信されます。このイベントは、新しい会議チェーン接続が追加されるたびに送信されます。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

すべてのスーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoConferenceChainAddedEv extends CiscoCallEv

フィールド

表 6-67　CiscoConferenceChainAddedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL,CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED,CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,

CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY,

META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-68　CiscoConferenceChainAddedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Connection	getAddedConnection()	直前にコールに追加された会議チェーン接続を返します。
CiscoConferenceChain	getConferenceChain()	チェーニングされているコールのすべての会議接続が含まれている CiscoConferenceChain を返します。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoConferenceChainRemovedEv

会議チェーン接続がコールから削除されると、CiscoConferenceChainRemovedEv イベントが送信されます。このイベントは、会議チェーン接続が削除されるたびに送信されます。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoConferenceChainRemovedEv extends CiscoCallEv

フィールド

表 6-69　CiscoConferenceChainRemovedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,

CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER,CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY,

META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-70　CiscoConferenceChainRemovedEf のメソッド

インターフェイス	メソッド	説明
CiscoConferenceChain	getConferenceChain()	チェーニングされているコールのすべての会議接続が含まれている CiscoConferenceChain を返します。戻り値：接続。
javax.telephony.Connection	getRemovedConnection()	直前にコールから削除された会議チェーン接続を返します。戻り値：CiscoConferenceChain。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoConferenceEndEv

CiscoConferenceEndEv イベントは、会議の動作が完了したことを示します。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoConferenceEndEv extends CiscoCallEv

フィールド

表 6-71　CiscoConferenceEndEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,

CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE, CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE, CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL, CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE, CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED, CAUSE_CTIPRECEDENCELEVELEXCEEDED, CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE, CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER, CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER, CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY,

META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-72　CiscoConferenceEndEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getConferenceControllerAddress()	現在このコール（最初のコール）の会議コントローラとして機能しているアドレスを返します。
javax.telephony.Call	getConferencedCall()	マージされたコールを返します。このコールは Call.INVALID の状態です。
javax.telephony.Call[]	getFailedCalls()	会議に参加できなかったコールのリストを返します。
javax.telephony.Call	getFinalCall()	会議の終了後にアクティブになっているコールを返します。
javax.telephony.TerminalConnection	getHeldConferenceController()	現在このコール（最後のコール）の会議コントローラとして機能している TerminalConnection を返します。これは、会議開始時の状態が HELD であった TerminalConnection です。会議コントローラが監視されていない場合、このメソッドは null または TerminalConnection を返します。
javax.telephony.TerminalConnection	getTalkingConferenceController()	現在このコール（最初のコール）の会議コントローラとして機能している TerminalConnection を返します。これは、当初 TALKING 状態だった TerminalConnection です。会議コントローラが監視されていない場合、このメソッドは null または TerminalConnection を返します。

表 6-72　CiscoConferenceEndEv のメソッド

インターフェイス	メソッド	説明
boolean	isSuccess()	会議 が正常に実行されたか失敗したかに応じて、ブール値（true または false）を返します。アプリケーションは、このインターフェイスを使用して、会議が正常に実行されたかどうかを確認できます。 会議が正常に実行されない原因として、次のような状況が考えられます。 • アプリケーションが Call.conference(otherCalls[]) 要求を発行し、1 つ以上のコールが会議に参加できなかった場合、この会議は失敗と見なされます。getFailedCalls() を使用して、失敗したコールを見つけます。 • 会議ブリッジがなく、会議が完了できなかった場合。getFailedCalls() を使用して、会議に参加できなかったコールのリストを取得します。 • 会議が開催できる前に会議に参加していた側がドロップした場合。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

isSuccess() も参照してください。

CiscoConferenceStartEv

CiscoConferenceStartEv イベントは、会議の動作が開始されたことを示します。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	回線をまたいで参加（Join Across Lines）/Connected Conference 機能に getControllerTerminalName() メソッドが追加されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoConferenceStartEv extends CiscoCallEv

フィールド

表 6-73　CiscoConferenceStartEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG, CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE, CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,

CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,
CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL,
CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS,
CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER,
CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR,
CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING,
CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION,
CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING,
CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL,
CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER,
CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR,
CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR,
CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL,
CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED,
CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY,
CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED,
CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT,
CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION,
CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE,
CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT,
CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE,

CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-74　CiscoConferenceStartEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getConferenceControllerAddress()	現在このコール（最初のコール）の会議コントローラとして機能しているアドレスを返します。
javax.telephony.Call	getConferencedCall()	会議に追加されるコールを返します。これは、会議の最初のコールへマージされるコールです。このインターフェイスは、会議に参加しているコールのリストから最初のコールを返します。
javax.telephony.Call[]	getConferencedCalls()	会議に追加されるコールのリストを返します。これらのコールは、会議の最後のコールへマージされるコールです。
javax.telephony.Call	getFinalCall()	会議の終了後にアクティブになっているコールを返します。このコールには、すべてのコールがマージされます。
javax.telephony.TerminalConnection	getHeldConferenceController()	現在このコール（最初のコール）の会議コントローラとして機能している TerminalConnection を返します。これは、当初 HELD 状態だった TerminalConnection です。会議コントローラが監視されていない場合、このメソッドは null を返します。複数コールを結合する場合、このメソッドは、最初の HELD 状態のコントローラを返します。
javax.telephony.TerminalConnection[]	getHeldConferenceControllers()	会議に参加していて、HELD 状態の会議コントローラ端末上のすべての TerminalConnections を返します。

表 6-74　CiscoConferenceStartEv のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Address	getOriginalConferenceController Address()	会議を開始した参加者のアドレスを返します。
javax.telephony.Terminal Connection	getTalkingConferenceController()	現在このコール（最初のコール）の会議コントローラとして機能している TerminalConnection を返します。これは、当初 TALKING 状態だった TerminalConnection です。会議コントローラが監視されていない場合、このメソッドは null を返します。TALKING 状態のコントローラがない場合、このメソッドは null を返します。TALKING 状態のコントローラがない会議にコールを参加させることができます。
String	getControllerTerminalName()	会議が行われるコントローラの端末名を返します。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoConnection

CiscoConnection インターフェイスは、Cisco 固有の機能を追加することによって CallControlConnection インターフェイスを拡張します。アプリケーションは getReason メソッドを使用して、接続が確立された原因を取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	追加された 2 つの新しいメソッド：getPartyInfo と任意の通話者のドロップ（Drop Any Party）機能の disconnect(CiscoPartyInfopartyInfo

すべてのスーパーインターフェイス

javax.telephony.callcontrol.CallControlConnection, CiscoObjectContainer, javax.telephony.Connection

宣言

public interface CiscoConnection extends javax.telephony.callcontrol.CallControlConnection, CiscoObjectContainer

フィールド

表 6-75　CiscoConnection のフィールド

インターフェイス	フィールド	説明
static int	ADDRESS_SEARCH_SPACE	リダイレクトは、リダイレクト コントローラのアドレスの検索スペースを使用して実行します。
static int	CALLED_ADDRESS_DEFAULT	Cisco JTAPI のデフォルト動作を適用します。
static int	CALLED_ADDRESS_SET_TO_PREFERRED CALLEDPARTY	元の着信アドレスは、preferredOriginalCalledParty フィールドに存在する値に設定します。
static int	CALLED_ADDRESS_SET_TO_REDIRECT_ DESTINATION	着側のアドレスをリダイレクト先にリセットします。
static int	CALLED_ADDRESS_UNCHANGED	リダイレクトの完了後も元の calledAddress は変更されません。
static int	CALLINGADDRESS_SEARCH_SPACE	リダイレクトは、発信元アドレスの検索スペースを使用して実行します。
static int	DEFAULT_SEARCH_SPACE	リダイレクトは、デフォルトの検索スペースを使用して実行します。
static int	REASON_DIRECTCALL	この接続は、直接コールの結果として確立されました。
static int	REASON_FORWARDALL	この接続は、無条件転送の結果として確立されました。

表 6-75　CiscoConnection のフィールド

インターフェイス	フィールド	説明
static int	REASON_FORWARDBUSY	この接続は、使用中での転送の結果として確立されました。
static int	REASON_FORWARDNOANSWER	この接続は、無応答時転送の結果として確立されました。
static int	REASON_OUTBOUND	この接続は、宛先側の接続ではなく発側の接続です。
static int	REASON_REDIRECT	この接続は、リダイレクションの結果として確立されました。
static int	REASON_TRANSFERREDCALL	この接続は、転送の結果として確立されました。
static int	REDIRECT_DROP_ON_FAILURE	このリダイレクト モードが指定されている場合は、宛先が有効であるかどうか、宛先が使用可能であるかどうかを確認せずにリダイレクトが実行されます。
static int	REDIRECT_NORMAL	このリダイレクト モードが指定されている場合は、宛先が有効かつ使用可能な場合にだけリダイレクトが実行されます。

継承したフィールド

インターフェイス javax.telephony.callcontrol.CallControlConnection から
ALERTING, DIALING, DISCONNECTED, ESTABLISHED, FAILED, IDLE, INITIATED, NETWORK_ALERTING, NETWORK_REACHED, OFFERED, OFFERING, QUEUED, UNKNOWN

インターフェイス javax.telephony.Connection から
CONNECTED, INPROGRESS

メソッド

表 6-76　CiscoConnection のメソッド

インターフェイス	メソッド	説明
Boolean	getAddressPI()	接続が作成されるアドレスに関連付けられた Presentation Indicator（PI）を返します。
CiscoConnectionID	getConnectionID()	この CiscoConnection の CiscoConnectionID を返します。
java.lang.String	getDParkPrefixCode()	コールを取得するために DPark DN と一緒にダイヤルする必要があるプレフィクス コードを返します。

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
int	getReason()	この接続が確立された原因を返します。エンドポイントで接続が確立された原因を識別しないと、適切に機能しないアプリケーションもあります。 接続の確立の原因を示す定数は、次のとおりです。 • CiscoConnection.REASON_DIRECTCALL CiscoConnection.REASON_TRANSFERREDCALL • CiscoConnection.REASON_FORWARDNOANSWER • CiscoConnection.REASON_FORWARDBUSY • CiscoConnection.REASON_FORWARDALL • CiscoConnection.REASON_REDIRECT • CiscoConnection.REASON_NORMAL
javax.telephony.TerminalConnection	getRequestController()	接続の現在の要求コントローラを返します。
java.lang.String	park()	このメソッドは、システム パーク DN にコールをパークして、その パーク DN のアドレスを返します。

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。 **例外** javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupported Exception, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailable Exception **パラメータ** Mode：パラメータには次の 2 つのいずれかの値を指定できます。 • CiscoConnection.REDIRECT_DROP_ON_FAILURE：このモードが指定されている場合は、宛先が有効であるかどうか、宛先が使用可能であるかどうかを確認せずにリダイレクトが実行されます。宛先が有効でない場合、または使用中の場合は、元のコールがドロップされます。 • CiscoConnection.REDIRECT_NORMAL：このモードが指定されている場合は、宛先が有効であるかどうか、宛先が使用可能であるかどうかを確認した後にだけリダイレクトが実行されます。これは CallControlConnection.redirect() メソッドの動作と同じです。失敗した場合でも、元のコールはドロップされません。

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。このメソッドは 2 種類の新しいパラメータ redirectMode および callingSearchSpace を受け付けます。 redirectMode は、実行するリダイレクトの種類を選択します。callingSearchSpace は、発側の検索スペースまたはリダイレクト コントローラの検索スペースのどちらかを使用する実装を指定します。 **パラメータ** mode：次のいずれかです。 • CiscoConnection.REDIRECT_DROP_ON_FAILURE：このモードが指定されている場合は、宛先が有効であるかどうか、宛先が使用可能であるかどうかを確認せずにリダイレクトが実行されます。宛先が有効でない場合、または使用中の場合は、元のコールがドロップされます。 • CiscoConnection.REDIRECT_NORMAL：このモードが指定されている場合は、宛先が有効であるかどうか、宛先が使用可能であるかどうかを確認した後にだけリダイレクトが実行されます。これは CallControlConnection.redirect() メソッドの動作と同じです。失敗した場合でも、元のコールはドロップされません。 callingSearchSpace：次のいずれかです。 • CiscoConnection.DEFAULT_SEARCH_SPACE • CiscoConnection.CALLINGADDRESS_SEARCH_SPACE • CiscoConnection.ADDRESS_SEARCH_SPACE **例外** javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupported Exception, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailable Exception

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。 このメソッドは 3 種類の新しいパラメータ mode、callingSearchSpace、および calledAddressOption を受け付けます。redirectMode は、実行するリダイレクトの種類を選択します。callingSearchSpace は、発側の検索スペースまたはリダイレクト コントローラの検索スペースのどちらかを使用する実装を指定します。calledAddressOption パラメータは、元の着信フィールドをリセットするかどうかを制御します。 **パラメータ** mode：次のいずれかです。 • CiscoConnection.REDIRECT_DROP_ON_FAILURE • CiscoConnection.REDIRECT_NORMAL callingSearchSpace：次のいずれかです。 • CiscoConnection.DEFAULT_SEARCH_SPACE • CiscoConnection.CALLINGADDRESS_SEARCH_SPACE • CiscoConnection.ADDRESS_SEARCH_SPACE calledAddressOption：次のいずれかです。 • CiscoConnection.CALLED_ADDRESS_DEFAULT • CiscoConnection.CALLED_ADDRESS_UNCHANGED • CiscoConnection.CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION **例外** javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupported Exception, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailable Exception

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, java.lang.String preferredOriginalCalledParty, java.lang.String facCode, java.lang.String cmcCode)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。このメソッドは 3 種類の新しいパラメータ mode、callingSearchSpace、および calledAddressOption を受け付けます。 redirectMode は、実行するリダイレクトの種類を選択します。callingSearchSpace は、発側の検索スペースまたはリダイレクト コントローラの検索スペースのどちらかを使用する実装を指定します。calledAddressOption パラメータは、元の着信フィールドをリセットするかどうかを制御します。 FAC コードおよび CMC コードが不明であるか無効な場合、コールが提供されない可能性や、platformException に次のエラー コードのいずれかが含まれる可能性があります。 • CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_NEEDED • CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_NEEDED • CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED • CiscoJTAPIException.CTIERR_FAC_CMC_REASON_FAC_INVALID • CiscoJTAPIException.CTIERR_FAC_CMC_REASON_CMC_INVALID **パラメータ** mode：次のいずれかです。 • CiscoConnection.REDIRECT_DROP_ON_FAILURE • CiscoConnection.REDIRECT_NORMAL callingSearchSpace：次のいずれかです。 • CiscoConnection.DEFAULT_SEARCH_SPACE • CiscoConnection.CALLINGADDRESS_SEARCH_SPACE • CiscoConnection.ADDRESS_SEARCH_SPACE

表 6-76　CiscoConnection のメソッド（続き）

インターフェイス	メソッド	説明
		• preferredOriginalCalledParty：destinationAddress に対してコールが提供されたときに originalCalledParty フィールドになる DN。このフィールド * を使用する必要がある場合、アプリケーションは calledAddressOption に CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY を設定する必要があります。アプリケーションがこのフィールドを指定していない場合、デフォルト値の null を渡す必要があります。 calledAddressOption：次のいずれかです。 • CiscoConnection.CALLED_ADDRESS_DEFAULT • CiscoConnection.CALLED_ADDRESS_UNCHANGED • CiscoConnection.CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION preferredOriginalCalledParty：destinationAddress に対してコールが提供されたときに originalCalledParty フィールドになる DN。このフィールド * を使用する必要がある場合、アプリケーションは calledAddressOption に CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY を設定する必要があります。アプリケーションがこのフィールドを指定していない場合、デフォルト値の null を渡す必要があります。 facCode：destinationAddress がコールをオファーするのに Forced Authorization Code（FAC）を要求する場合に必要です。このパラメータで FAC を渡します。destinationAddress が FAC コードを要求しない場合、デフォルト値の null を渡します。 cmcCode：destinationAddress がコールをオファーするのに Client Matter Code（CMC）を要求する場合に必要です。このパラメータで CMC を渡します。destinationAddress が CMC コードを要求しない場合、デフォルト値の null を渡します。

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, int calledAddressOption, java.lang.String preferredOriginalCalledParty, java.lang.String facCode, java.lang.String cmcCode, int featurePriority)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。これは新しいパラメータ、featurePriority をオーバーロードします。コール優先度を設定します。 featurePriority パラメータは、次のいずれかになります。 • CiscoCall.FEATUREPRIORITY_NORMAL • CiscoCall.FEATUREPRIORITY_URGENT • CiscoCall.FEATUREPRIORITY_ EMERGENCY **戻り値** Connection **例外** javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupported Exception, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailable Exception

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection	redirect(java.lang.String destinationAddress, int mode, int callingSearchSpace, java.lang.String preferredOriginalCalledParty)	このメソッドは CallControlConnection.redirect() メソッドをオーバーロードします。このメソッドは 3 種類の新しいパラメータ mode、callingSearchSpace、および preferredOriginalCalledParty を受け付けます。 redirectMode は、実行するリダイレクトの種類を選択します。callingSearchSpace は、発側の検索スペースまたはリダイレクト コントローラの検索スペースのどちらかを使用する実装を指定します。 **パラメータ** mode：次のいずれかです。 • CiscoConnection.REDIRECT_DROP_ON_FAILURE • CiscoConnection.REDIRECT_NORMAL callingSearchSpace：次のいずれかです。 • CiscoConnection.DEFAULT_SEARCH_SPACE • CiscoConnection.CALLINGADDRESS_SEARCH_SPACE • CiscoConnection.ADDRESS_SEARCH_SPACE preferredOriginalCalledParty：destinationAddress に対してコールが提供されたときに originalCalledParty フィールドになる DN。 **例外** javax.telephony.InvalidStateException, javax.telephony.InvalidPartyException, javax.telephony.MethodNotSupported Exception, javax.telephony.PrivilegeViolationException, javax.telephony.ResourceUnavailable Exception
void	setRequestController(javax.telephony.TerminalConnection tc)	このインターフェイスは、要求側の TerminalConnection に対して提供されます。
com.cisco.jtapi.extensions.CiscoPartyInfo[]	getPartyInfo()	参加者のリストを返します。

表 6-76　CiscoConnection のメソッド （続き）

インターフェイス	メソッド	説明
java.lang.void	disconnect(CiscoPartyInfoparty Info)	渡されたパラメータ値と CiscoPartyInfo が一致している参加者を接続解除し、一致しない場合は例外をスローします。 **例外** PrivilegeViolationException, InvalidStateException

継承したメソッド

インターフェイス javax.telephony.callcontrol.CallControlConnection から
accept, addToAddress, getCallControlState, park, redirect, reject

インターフェイス javax.telephony.Connection から
disconnect, getAddress, getCall, getCapabilities, getConnectionCapabilities, getState, getTerminalConnections

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

資料

なし

CiscoConnectionID

CiscoConnectionID オブジェクトは、各接続に関連付けられる一意のオブジェクトです。A アプリケーションでは、オブジェクト自体または intValue() メソッドで返されたオブジェクトの整数表現を使用できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoObjectContainer

宣言

public interface CiscoConnectionID extends CiscoObjectContainer

フィールド

なし

メソッド

表 6-77　CiscoConnectionID のメソッド

インターフェイス	メソッド	説明
CiscoConnection	getConnection()	CiscoConnectionID の CiscoConnection を返します。
Int	intValue()	このオブジェクトの整数表現を返します。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

なし

CiscoConsultCall

CiscoConsultCall インターフェイスは CiscoCall インターフェイスを拡張し、コンサルト転送またはコンサルト会議の一部として作成されたコールのプロパティへのアクセスを可能にします。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.Call, javax.telephony.callcontrol.CallControlCall, CiscoCall, CiscoObjectContainer

宣言

public interface CiscoConsultCall extends CiscoCall

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCall から
CALLSECURITY_AUTHENTICATED, CALLSECURITY_ENCRYPTED, CALLSECURITY_NOTAUTHENTICATED, CALLSECURITY_UNKNOWN, FEATUREPRIORITY_EMERGENCY, FEATUREPRIORITY_NORMAL, FEATUREPRIORITY_URGENT, PLAYTONE_BOTHLOCALANDREMOTE, PLAYTONE_LOCALONLY, PLAYTONE_NOLOCAL_OR_REMOTE, PLAYTONE_REMOTEONLY, SILENT_MONITOR

インターフェイス javax.telephony.Call から
ACTIVE, IDLE, INVALID

メソッド

表 6-78　CiscoConsultCall のメソッド

インターフェイス	メソッド	説明
javax.telephony.Terminal Connection	getConsultingTerminalConnection()	この CiscoConsultCall の作成に使用されたコンサルティング TerminalConnection を返します。このコールがコンサルト転送またはコンサルト会議の一部として作成されていた場合、元のコールのコンサルテーションを実行するのに使用された TerminalConnection が getConsultingTerminalConnection メソッドによって返されます。このメソッドでは、ConsultCall と最初のコールを相互に関連付けることができます。最初のコールは、自身に関連付けられている ConsultCall を識別するためのメソッドを持っていません。戻り値：このコールがコンサルテーションの結果作成されたものでない場合は null、このコールがコンサルテーションの結果作成されたものである場合は、元のコールのコンサルティング TerminalConnection。

表 6-78　CiscoConsultCall のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection[]	consultWithoutMedia (javax.telephony.TerminalConnection tc, java.lang.String dialedDigits)	メディアを設定することなく、コンサルト コールを開始する機能をアプリケーションに提供します。このインターフェイスは、アプリケーションがコンサルト コールを作成し、コンサルト コールのメディアが確立する前に転送が完了する場合に起動できます。 コンサルト コールが応答され、コンサルト コールの設定中にアプリケーションが転送を完了した場合、Cisco Unified Communication Manager は誤りのある競合状態で稼動する可能性があります。 この問題を回避するために、コンサルト コールのメディアの設定を待機しないアプリケーションがこのメソッドを使用して、コンサルト コールを設定することができます。 CallEvent の観点からすると、このメソッドの動作は CallControlCall.consult(TerminalConnection tc, String dialedDigits) に類似しています。 このメソッドは、この Call と他のアクティブ Call の間のコンサルテーションを作成します。ただし、メディアは確立されません。このコンサルト コールを転送することは可能ですが、会議に追加できません。Cisco JTAPI では、CallControlCall.setConferenceEnable() でこのメソッドをサポートしません。Cisco JTAPI では、CallControlCall.setTransferEnable() でだけこのメソッドをサポートします。 **例外** javax.telephony.InvalidStateException javax.telephony.InvalidArgumentException javax.telephony.MethodNotSupported Exception javax.telephony.ResourceUnavailableException javax.telephony.PrivilegeViolationException javax.telephony.InvalidPartyException

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCall から

conference, connect, getCalledAddressPI, getCalledPartyInfo, getCallID, getCallingAddressPI, getCallSecurityStatus, getConferenceChain, getCurrentCalledAddress, getCurrentCalledAddressPI, getCurrentCalledDisplayNamePI, getCurrentCalledPartyDisplayName, getCurrentCalledPartyInfo, getCurrentCalledPartyUnicodeDisplayName, getCurrentCalledPartyUnicodeDisplayNamelocale, getCurrentCallingAddress, getCurrentCallingAddressPI, getCurrentCallingDisplayNamePI,

getCurrentCallingPartyDisplayName, getCurrentCallingPartyInfo, getCurrentCallingPartyUnicodeDisplayName, getCurrentCallingPartyUnicodeDisplayNamelocale, getGlobalizedCallingParty, getLastRedirectedPartyInfo, getLastRedirectingAddressPI, getLastRedirectingPartyInfo, getModifiedCalledAddress, getModifiedCallingAddress, startMonitor, startMonitor, transfer

インターフェイス javax.telephony.callcontrol.CallControlCall から
addParty, conference, consult, consult, drop, getCalledAddress, getCallingAddress, getCallingTerminal, getConferenceController, getConferenceEnable, getLastRedirectedAddress, getTransferController, getTransferEnable, offHook, setConferenceController, setConferenceEnable, setTransferController, setTransferEnable, transfer, transfer

インターフェイス javax.telephony.Call から
addObserver, connect, getCallCapabilities, getCapabilities, getConnections, getObservers, getProvider, getState, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

詳細については、CiscoCall を参照してください。

CiscoConsultCallActiveEv

CiscoConsultCallActiveEv イベント インターフェイスは JTAPI の CallActiveEv イベントを拡張します。このイベントは、コール オブジェクトの状態が Call.ACTIVE に変わり、（手動またはプログラムによる）コンサルト転送動作またはコンサルト会議動作の結果としてコールが発信されたことを示します。アプリケーションは CiscoConsultCall.getConsultingTerminalConnection メソッドを使用して、最初の（コンサルト）コールのコンサルティング TerminalConnection を取得できます。

このイベントは CallObserver インターフェイスによってアプリケーションに報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallActiveEv, javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoConsultCallActiveEv extends CiscoCallEv, javax.telephony.events.CallActiveEv

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から

CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,
CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL,
CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,

CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER,CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

javax.telephony.TerminalConnectiongetHeldTerminalConnection() 推奨されません。CiscoConsultCall.getConsultingTerminalConnection() に置き換えられました。

表 6-79　CiscoConsultCallActiveEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.TerminalConnection	getHeldTerminalConnection()	**推奨されないメソッドです。** CiscoConsultCall.getConsultingTerminal Connection() に置き換えられました。 この CiscoConsultCall の作成に使用されたコンサルティング TerminalConnection を返します。このメソッドを使用して、コンサルト コールと最初のコールを相互に関連付けることができます。最初のコールは、自身に関連付けられているコンサルト コールを識別するためのメソッドを持っていません。戻り値：このイベントに関連付けられているコールを作成したコールのコンサルティング TerminalConnection。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、Call、CallObserver、CallActiveEv と「定数フィールド値」（P.F-1）を参照してください。

CiscoEv

このコードの JTAPI javax.telephony.events.Ev インターフェイスを拡張する CiscoEv インターフェイスは、Cisco によって拡張されたすべての JTAPI イベントの基本インターフェイスになります。このパッケージのイベントはすべて、直接的または間接的にこのインターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.Ev

サブインターフェイス

CiscoAddrActivatedEv, CiscoAddrActivatedOnTerminalEv, CiscoAddrAddedToTerminalEv, CiscoAddrAutoAcceptStatusChangedEv, CiscoAddrCreatedEv, CiscoAddrEv, CiscoAddrInServiceEv, CiscoAddrIntercomInfoChangedEv, CiscoAddrIntercomInfoRestorationFailedEv, CiscoAddrOutOfServiceEv, CiscoAddrRecordingConfigChangedEv, CiscoAddrRemovedEv, CiscoAddrRemovedFromTerminalEv, CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv, CiscoCallChangedEv, CiscoCallEv, CiscoCallSecurityStatusChangedEv, CiscoConferenceChainAddedEv, CiscoConferenceChainRemovedEv, CiscoConferenceEndEv, CiscoConferenceStartEv, CiscoConsultCallActiveEv, CiscoMediaOpenLogicalChannelEv, CiscoOutOfServiceEv, CiscoProvCallParkEv, CiscoProvEv, CiscoProvFeatureEv, CiscoProvTerminalCapabilityChangedEv, CiscoRestrictedEv, CiscoRTPInputKeyEv, CiscoRTPInputStartedEv, CiscoRTPInputStoppedEv, CiscoRTPOutputKeyEv, CiscoRTPOutputStartedEv, CiscoRTPOutputStoppedEv, CiscoTermActivatedEv, CiscoTermButtonPressedEv, CiscoTermCreatedEv, CiscoTermDataEv, CiscoTermDeviceStateActiveEv, CiscoTermDeviceStateAlertingEv, CiscoTermDeviceStateHeldEv, CiscoTermDeviceStateIdleEv, CiscoTermDeviceStateWhisperEv, CiscoTermDNDOptionChangedEv, CiscoTermDNDStatusChangedEv, CiscoTermEv, CiscoTermInServiceEv, CiscoTermOutOfServiceEv, CiscoTermRegistrationFailedEv, CiscoTermRemovedEv, CiscoTermRestrictedEv, CiscoTermSnapshotCompletedEv, CiscoTermSnapshotEv, CiscoToneChangedEv, CiscoTransferEndEv, CiscoTransferStartEv

宣言

public interface CiscoEv extends javax.telephony.events.Ev

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、Ev を参照してください。

CiscoFeatureReason

CiscoFeatureReason インターフェイスは、配信される各イベントに関連付けられた機能原因を指定します。

インターフェイス履歴

Cisco Unified Communications Manager リリース	説明
7.1(1 および 2)	パーク モニタリングおよび Assisted DPark 機能のための新しい原因、FORWARD_NO_RETRIEVE が追加されました。

宣言

public interface CiscoFeatureReason

フィールド

表 6-80　CiscoFeatureReason のフィールド

インターフェイス	フィールド	説明
static int	REASON_BARGE	イベントに関する理由が割り込み機能であることを示します。
static int	REASON_BLINDTRANSFER	理由がシングル ステップ転送であることを示します。
static int	REASON_CALLPICKUP	イベントに関する理由がピックアップであることを示します。
static int	REASON_CCM_REDIRECTION	イベントに関する理由が SIP 3xx 機能であることを示します。
static int	REASON_CLICK_TO_CONFERENCE	クリック ツー会議機能を使用して接続が作成されたか削除されたことを示します。
static int	REASON_CONFERENCE	イベントに関する理由が会議であることを示します。
static int	REASON_DPARK_CALLPARK	イベントに関する理由が DPARK 機能であることを示します。
static int	REASON_DPARK_REVERSION	イベントに関する理由が DPARK の復帰であることを示します。
static int	REASON_DPARK_UNPARK	イベントに関する理由が DPARK のパーク解除であることを示します。
static int	REASON_FAC_CMC	イベントに関する理由が FAC 機能、CMC 機能であることを示します。
static int	REASON_FORWARDALL	イベントに関する理由が転送であることを示します。
static int	REASON_FORWARDBUSY	イベントに関する理由が話中転送であることを示します。
static int	REASON_FORWARDNOANSWER	イベントに関する理由が無応答時転送であることを示します。
static int	REASON_FORWARD_NO_RETRIEVE	イベントに関する理由が取得転送なしであることを示します。
static int	REASON_IMMDIVERT	イベントに関する理由が即時転送であることを示します。
static int	REASON_MOBILITY	イベントに関する理由が Mobility Manager 機能であることを示します。
static int	REASON_MOBILITY_CELLPICKUP	イベントに関する理由が Mobility Manager 機能であることを示します。
static int	REASON_MOBILITY_FOLLOWME	イベントに関する理由が Mobility Manager 機能であることを示します。

表 6-80　CiscoFeatureReason のフィールド

インターフェイス	フィールド	説明
static int	REASON_MOBILITY_HANDIN	イベントに関する理由が Mobility Manager 機能であることを示します。
static int	REASON_MOBILITY_HANDOUT	イベントに関する理由が Mobility Manager 機能であることを示します。
static int	REASON_MOBILITY_IVR	イベントに関する理由が Mobility Manager 機能であることを示します。
static int	REASON_NORMAL	イベントに関する理由が通常であることを示します。
static int	REASON_PARK	イベントに関する理由がパーク機能であることを示します。
static int	REASON_PARKREMAINDER	イベントに関する理由がパーク リマインダであることを示します。
static int	REASON_QSIG_PR	イベントに関する理由が QSIG パス置換であることを示します。
static int	REASON_REDIRECT	イベントに関する理由がリダイレクトであることを示します。
static int	REASON_REFER	Cisco Unified Communications Manager で参照が実行されるために送信されるイベントに返されます。
static int	REASON_REPLACE	REASON_REPLACE：この理由は Cisco Unified Communications Manager で置換機能が実行されるために送信されるイベントに返されます。
static int	REASON_SILENTMONITORING	イベントに関する理由がサイレント モニタリングであることを示します。
static int	REASON_TRANSFER	イベントに関する理由が転送であることを示します。
static int	REASON_UNPARK	イベントに関する理由がパーク解除であることを示します。

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoIntercomAddress

CiscoIntercomAddress インターフェイスは、インターコム アドレス用に Unified CM 固有の機能を追加することによって CiscoAddress インターフェイスを拡張します。このインターフェイスを使用すると、アプリケーションからインターコム コールを開始したり、他のインターコム固有の機能を利用できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.Address, CiscoAddress, CiscoObjectContainer

宣言

public interface CiscoIntercomAddress extends CiscoAddress

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoAddress から
APPLICATION_CONTROLLED_RECORDING, AUTO_RECORDING, AUTOACCEPT_OFF, AUTOACCEPT_ON, AUTOANSWER_OFF, AUTOANSWER_UNKNOWN, AUTOANSWER_WITHHEADSET, AUTOANSWER_WITHSPEAKERSET, EXTERNAL, EXTERNAL_UNKNOWN, IN_SERVICE, INTERNAL, MONITORING_TARGET, NO_RECORDING, OUT_OF_SERVICE, RINGER_DEFAULT, RINGER_DISABLE, RINGER_ENABLE, UNKNOWN

メソッド

表 6-81　CiscoIntercomAddress のメソッド

インターフェイス	メソッド	説明
void	setIntercomTarget(java.lang.String targetDNjava.lang.String targetAsciiLabel, java.lang.String targetUnicodeLabel)	電話機のインターコム回線の横に表示されるインターコム ターゲット DN、インターコム ターゲット ラベルおよびインターコム ターゲット Unicode ラベルを設定します。電話機に Unicode ラベル機能がある場合には Unicode ラベルが表示され、そうでない場合は、ASCII ターゲット ラベルが表示されます。 **例外** javax.telephony.InvalidPartyException はターゲット DN が無効であることを意味します。 javax.telephony.InvalidStateException は、アドレス、端末、またはプロバイダーがイン サービスではないことを意味します。 **パラメータ** • targetDN：インターコム コールの宛先 DN • targetAsciiLabel：電話機ターゲットのインターコム回線の横に表示される ASCII 表示ラベル • UnicodeLabel：電話機に表示される Unicode 表示ラベル
Boolean	isIntercomTargetSet()	アプリケーションが現在の値をオーバーライドした場合は true を返し、現在の値がデータベースで設定されたデフォルト値と一致する場合は false を返します。
void	resetIntercomTarget()	インターコム ターゲット DN、インターコム ターゲット ラベルおよびインターコム ターゲット Unicode ラベルをデフォルト値にリセットします。 **例外** javax.telephony.InvalidPartyException, javax.telephony.InvalidStateException
java.lang.String	getIntercomTargetNumber()	アプリケーションで設定される現在のインターコム ターゲット DN を返します。アプリケーションがインターコム ターゲット DN を設定していない場合、このインターフェイスは Cisco Unified CM Administration で設定されるデフォルトのインターコム ターゲット DN を返します。戻り値：インターコム ターゲットの DN 番号（文字列）。

表 6-81　CiscoIntercomAddress のメソッド （続き）

インターフェイス	メソッド	説明
java.lang.String	getIntercomTargetAsciiLabel()	アプリケーションで設定される現在のインターコム ターゲット ラベルを返します。アプリケーションがインターコム ターゲット ラベルを設定していない場合、このインターフェイスは Cisco Unified CM Administration で設定されるデフォルトのインターコム ターゲット ラベルを返します。戻り値 : インターコム ターゲットのラベル文字列。
java.lang.String	getIntercomTargetUnicodeLabel()	アプリケーションで設定される現在のインターコム ターゲット Unicode ラベルを返します。アプリケーションが Unicode ラベルを設定していない場合、このインターフェイスは Cisco Unified CM Administration で設定されるデフォルトのインターコム ターゲット Unicode ラベルを返します。戻り値 : インターコム Unicode ターゲットのラベル文字列。
java.lang.String	getDefaultIntercomTargetNumber()	Cisco Unified CM Administration から設定されるデフォルトのインターコム ターゲット DN を返します。戻り値 : デフォルトのインターコム ターゲットの DN 番号（文字列）。
java.lang.String	getDefaultIntercomTargetAsciiLabel()	Cisco Unified CM Administration から設定されるデフォルトのインターコム ターゲット ラベルを返します。戻り値 : デフォルトのインターコム ターゲットのラベル文字列。
java.lang.String	getDefaultIntercomTargetUnicodeLabel()	Cisco Unified CM Administration から設定されるデフォルトのインターコム ターゲット ラベルを返します。戻り値 : デフォルトの Unicode インターコム ターゲットのラベル文字列。

表 6-81　CiscoIntercomAddress のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony.Connection[]	connectIntercom(javax.telephony.Terminal terminal, java.lang.String targetNumber)	発信側インターコム アドレスから着信側インターコム アドレスにインターコム コールをかけます。戻り値：発側と着側のインターコム アドレスの接続リスト。 **例外** javax.telephony.InvalidPartyException：ターゲット DN は有効な番号ではありません。 javax.telephony.InvalidArgumentException：アドレスが CiscoIntercomAddress でないか、または端末が Terminal ではありません。 javax.telephony.InvalidStateException ：アドレス、端末、またはプロバイダーがイン サービスではありません。 javax.telephony.ResourceUnavailableException：オペレーションを完了するためのリソースを利用できません。 javax.telephony.PrivilegeViolationException：アプリケーションにこのオペレーションを実行するための権限がありません。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoAddress から

clearCallConnections, getAddressCallInfo, getAutoAcceptStatus, getAutoAnswerStatus, getInServiceAddrTerminals, getPartition, getRecordingConfig, getRegistrationState, getRestrictedAddrTerminals, getState, getType, isRestricted, setAutoAcceptStatus, setMessageWaiting, setRingerStatus

インターフェイス javax.telephony.Address から

addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から

getObject, setObject

関連資料

詳細については、CiscoAddress を参照してください。

CiscoJtapiException

CiscoJtapiException インターフェイスは CTI 要求を返すエラー コードを定義します。このインターフェイスを実装するために、Cisco JTAPI は すべての JTAPI 例外を拡張しています。例外を CiscoJtapiException にキャストし、メソッド getErrorCode() を呼び出すことによってエラー コードを取得できます。

たとえば、「e」がアプリケーションで検出された例外の場合は、次のコードでその例外が CiscoJtapiException のインスタンスかどうかを確かめます。

```
try {
   // some code here
 }
catch ( Exception e ) {
   if( e instanceof CiscoJtapiException){
     CiscoJtapiException ce = com.cisco.cti.client.CTIFAILURE.(CiscoJtapiException) e
     int errorCode = com.cisco.cti.client.CTIFAILURE.ce.getErrorCode()   //returns the
ErrorCode.
   }
 }
```

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.0	このインターフェイスが追加されました。
7.1(1 および 2)	論理パーティション設定機能の新しいエラー コード、CiscoJtapiException.CTIERR_REDIRECT_CALL_PARTITIONING_POLICY と CiscoJtapiException.CTIERR_FEATURE_NOT_AVAILABLE が追加されました。

宣言

public interface CiscoJtapiException

フィールド

表 6-82　CiscoJtapiException のフィールド

インターフェイス	フィールド	説明
static int	ASSOCIATED_LINE_NOT_OPEN	このエラーは、オープンされていない回線上で要求が発行されたことを示します。
static int	CALL_ALREADY_EXISTS	このエラーは、回線上にすでに別のコールが存在していることを示します。
static int	CALL_DROPPED	機能（保留、保留解除、転送、または会議）要求の後、要求が完了する前にコールがドロップされました。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CALLHANDLE_NOTINCOMINGCALL	このエラーは、存在しないか、または正しい状態にないコールに応答しようとしたことを示します。
static int	CALLHANDLE_UNKNOWN_TO_LINECONTROL	このエラーは、回線制御側で検知されていないコールをリダイレクトしようとしたことを示します。
static int	CANNOT_OPEN_DEVICE	このエラーは、関連するデバイスが登録解除のため、デバイスのオープンに失敗したことを示します。
static int	CANNOT_TERMINATE_MEDIA_ON_PHONE	このエラーは、デバイスに実際の電話がある場合に、メディアを終端できない（常に、電話がメディアを終端する）ことを示します。
static int	CFWDALL_ALREADY_SET	このエラーは、すでにオンになっている CFWALL をオンにしようとしたことを示します。
static int	CFWDALL_DESTN_INVALID	このエラーは、無効な宛先に CFWALL を実行しようとしたことを示します。
static int	CLUSTER_LINK_FAILURE	このエラーは、クラスタ内にある Cisco Unified CM の 1 つへのリンクが失敗したことを示します（ネットワーク エラー）。
static int	COMMAND_NOT_IMPLEMENTED_ON_DEVICE	このエラーは、デバイスがコマンドをサポートしていないことを示します。
static int	CONFERENCE_ALREADY_PRESENT	このエラーは、すでに会議に参加している通話者に会議を実行しようとしたことを示します。
static int	CONFERENCE_FAILED	このエラーは、会議の開催が成功しなかったことを示します。
static int	CONFERENCE_FULL	このエラーは、すべての会議ブリッジが使用中であることを示します。
static int	CONFERENCE_INACTIVE	このエラーは、コンサルト会議が有効ではないときに、会議を開催しようとしたことを示します。
static int	CONFERENCE_INVALID_PARTICIPANT	このエラーは、自分自身または無効な関係者に会議を実行しようとしたことを示します。
static int	CTIERR_ACCESS_TO_DEVICE_DENIED	このエラーは、デバイスへのアクセスが拒否されたことを示します。
static int	CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	このエラーは、アプリケーションのソフトキーが別のアプリケーションによってすでに制御されていることを示します。
static int	CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	このエラーは、アプリケーションのデータ サイズが限度を超えていることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_BIB_NOT_CONFIGURED	このエラーは、ビルトインブリッジ（BIB）が構成されていないことを示します。
static int	CTIERR_BIB_RESOURCE_NOT_AVAILABLE	このエラーは、ビルトインブリッジ（BIB）リソースが利用できないことを示します。
static int	CTIERR_CALL_MANAGER_NOT_AVAILABLE	このエラーは、Communications Manager が現在利用できないことを示します。
static int	CTIERR_CALL_NOT_EXISTED	このエラーは、コールが存在していないことを示します。
static int	CTIERR_CALL_PARK_NO_DN	このエラーは、コール パーク DN がないことを示します。
static int	CTIERR_CALL_REQUEST_ALREADY_OUTSTANDING	このエラーは、コール要求が未処理であることを示します。
static int	CTIERR_CALL_UNPARK_FAILED	このエラーは、コールのパーク解除が失敗したことを示します。
static int	CTIERR_CAPABILITIES_DO_NOT_MATCH	このエラーは、機能が適合しないことを示します。
static int	CTIERR_CLOSE_DELAY_NOT_SUPPORTED_WITH_REG_TYPE	このエラーは、この登録タイプではクローズ遅延がサポートされないことを示します。
static int	CTIERR_CONFERENCE_ALREADY_EXISTED	このエラーは、会議がすでに存在していることを示します。
static int	CTIERR_CONFERENCE_NOT_EXISTED	このエラーは、会議が存在していないことを示します。
static int	CTIERR_CONNECTION_ON_INVALID_PORT	このエラーは、アプリケーションが無効なポートに接続しようとしていることを示します。
static int	CTIERR_CONSULT_CALL_FAILURE	このエラーは、コンサルト コールが失敗したことを示します。
static int	CTIERR_CONSULTCALL_ALREADY_OUTSTANDING	このエラーは、コンサルト コールが未処理であることを示します。
static int	CTIERR_CRYPTO_CAPABILITY_MISMATCH	このエラーは、デバイスの暗号アルゴリズムが現在のデバイス登録と適合してないためにデバイス登録が失敗ことを示します。
static int	CTIERR_CTIHANDLER_PROCESS_CREATION_FAILED	このエラーは、CTIHandler プロセスの作成が失敗したことを示します。
static int	CTIERR_DB_INITIALIZATION_ERROR	このエラーは、DB 初期化エラーを示します。
static int	CTIERR_DEVICE_ALREADY_OPENED	このエラーは、デバイスがすでにオープンされていることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_DEVICE_NOT_OPENED_YET	このエラーは、デバイスがまだオープンされていないことを示します。
static int	CTIERR_DEVICE_OWNER_ALIVE_TIMER_STARTED	このエラーは、デバイスの登録に失敗したことを示します。
static int	CTIERR_DEVICE_REGISTRATION_FAILED_NOT_SUPPORTED_MEDIATYPE	このエラーは、無効なメディア タイプであることを示します。インターコム回線の場合、CTIPort はダイナミック メディア ポート登録で登録する必要があります。
static int	CTIERR_DEVICE_RESTRICTED	このエラーは、デバイスが制限されていることを示します。
static int	CTIERR_DEVICE_SHUTTING_DOWN	このエラーは、デバイスがシャット ダウン中であることを示します。
static int	CTIERR_DIRECTORY_LOGIN_TIMEOUT	このエラーは、ディレクトリのログインがタイム アウトになっていることを示します。
static int	CTIERR_DUPLICATE_CALL_REFERENCE	このエラーは、コールの参照値が重複していることを示します。
static int	CTIERR_DYNREG_IPADDRMODE_MISMATCH	これは、シスコのメディア/ルート端末が複数のアドレッシング モードで登録されている場合に登録に失敗したことを示します。
static int	CTIERR_FAC_CMC_REASON_CMC_INVALID	入力した Client Matter Code (CMC) が無効です。
static int	CTIERR_FAC_CMC_REASON_CMC_NEEDED	コールをオファーするのに CMC が必要です。
static int	CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	コールをオファーするのに Forced Authorization Code（FAC）および CMC が必要です。
static int	CTIERR_FAC_CMC_REASON_FAC_INVALID	入力された FAC が無効です。
static int	CTIERR_FAC_CMC_REASON_FAC_NEEDED	コールをオファーするのに FAC が必要です。
static int	CTIERR_FEATURE_ALREADY_REGISTERED	このエラーは、機能がすでに登録されていることを示します。
static int	CTIERR_FEATURE_DATA_REJECT	このエラーは、機能データが拒否されたことを示します。
static int	CTIERR_FEATURE_SELECT_FAILED	このエラーは、機能の選択に失敗したことを示します。
static int	CTIERR_ILLEGAL_DEVICE_TYPE	このエラーは、デバイス タイプが正しくないことを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_INCOMPATIBLE_AUTOINSTALL_PROTOCOL_VERSION	このエラーは、自動インストール プロトコルのバージョンに互換性がないことを示します。
static int	CTIERR_INCORRECT_MEDIA_CAPABILITY	不正なメディア機能が原因で、デバイス登録が失敗しました。
static int	CTIERR_INFORMATION_NOT_AVAILABLE	このエラーは、情報がないことを示します。
static int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_CONFIGURED	このエラーは、インターコム ターゲットの値がすでに構成され、アプリケーションがインターコム ターゲット DN でコールを開始しようとしていることを示します。
static int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_SET	このエラーは、インターコム ターゲットの値がすでに設定されており、アプリケーションが設定し直そうとしているためにインターコム要求が失敗したことを示します。
static int	CTIERR_INTERCOM_SPEEDDIAL_DESTN_INVALID	このエラーは、インターコム ターゲットの値がインターコム グループ内にないためにインターコム要求が失敗したことを示します。
static int	CTIERR_INTERCOM_TALKBACK_ALREADY_PENDING	このエラーは、インターコム応答要求がすでに処理中であることを示します。
static int	CTIERR_INTERCOM_TALKBACK_FAILURE	このエラーは、何らかの理由で応答要求が失敗したことを示します。
static int	CTIERR_INTERNAL_FAILURE	このエラーは、CTI 内部エラーが発生したことを示します。
static int	CTIERR_INVALID_CALLID	このエラーは、コール ID が無効であることを示します。
static int	CTIERR_INVALID_DEVICE_NAME	このエラーは、デバイス名が無効であることを示します。
static int	CTIERR_INVALID_DTMFDIGITS	Play DTMF 要求が無効な番号であるために失敗しました。
static int	CTIERR_INVALID_FILTER_SIZE	このエラーは、フィルタ サイズが無効であることを示します。
static int	CTIERR_INVALID_MEDIA_DEVICE	このエラーは、メディア デバイスが無効であることを示します。
static int	CTIERR_INVALID_MEDIA_PARAMETER	このエラーは、メディア パラメータが無効であることを示します。
static int	CTIERR_INVALID_MEDIA_PROCESS	このエラーは、無効なメディア プロセスが存在していることを示します。
static int	CTIERR_INVALID_MEDIA_RESOURCE_ID	このエラーは、メディア リソース ID が無効であることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_INVALID_MESSAGE_HEADER_INFO	このエラーは、ヘッダー情報が無効であることを示します。
static int	CTIERR_INVALID_MESSAGE_LENGTH	このエラーは、メッセージの長さが無効であることを示します。
static int	CTIERR_INVALID_MONITOR_DESTN	このエラーは、無効なモニタリング対象のためにモニタリング要求が失敗したことを示します。
static int	CTIERR_INVALID_MONITOR_DN_TYPE	このエラーは、モニタリング DN タイプが無効であることを示します。
static int	CTIERR_INVALID_MONITORMODE	このエラーは、モニタリングモードが無効であるためにモニタリング要求が失敗したことを示します。
static int	CTIERR_INVALID_PARAMETER	このエラーは、パラメータが無効であることを示します。
static int	CTIERR_INVALID_PARK_DN	このエラーは、DN が 無効なパーク DN であることを示します。
static int	CTIERR_INVALID_PARK_REGISTRATION_HANDLE	このエラーは、ハンドルが無効なパーク登録ハンドルであることを示します。
static int	CTIERR_INVALID_RESOURCE_TYPE	このエラーは、リソース タイプが無効であることを示します。
static int	CTIERR_IPADDRMODE_MISMATCH	これは、IP アドレッシング モードが一致しないために登録に失敗したことを示します。
static int	CTIERR_LINE_OUT_OF_SERVICE	このエラーは、回線がアウトオブサービスであることを示します。
static int	CTIERR_LINE_RESTRICTED	このエラーは、回線が制限されていることを示します。
static int	CTIERR_MAXCALL_LIMIT_REACHED	このエラーは、最大コール制限値に達したことを示します。
static int	CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC	このエラーは、デバイスを動的メディア終端で登録しようとしたためにデバイスの登録が失敗したことを示します。
static int	CTIERR_MEDIA_ALREADY_TERMINATED_NONE	このエラーは、デバイスがすでにメディア終端がなしで登録されているためにデバイスの登録が失敗したことを示します。
static int	CTIERR_MEDIA_ALREADY_TERMINATED_STATIC	このエラーは、デバイスを静的メディア終端で登録しようとしたためにデバイスの登録が失敗したことを示します。
static int	CTIERR_MEDIA_CAPABILITY_MISMATCH	このエラーは、デバイスのメディア機能が現在のデバイス登録と適合してないためにデバイス登録が失敗したことを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_MEDIA_RESOURCE_NAME_SIZE_EXCEEDED	このエラーは、メディア リソース名のサイズが限度を超えていることを示します。
static int	CTIERR_MEDIAREGISTRATIONTYPE_DO_NOT_MATCH	このエラーは、メディア登録のタイプが一致していないことを示します。
static int	CTIERR_MESSAGE_TOO_BIG	このエラーは、メッセージが長すぎることを示します。
static int	CTIERR_MORE_ACTIVE_CALLS_THAN_RESERVED	このエラーは、アクティブなコールが予約されたコールよりも多いことを示します。
static int	CTIERR_NO_EXISTING_CALLS	このエラーは、コールが存在していないことを示します。
static int	CTIERR_NO_EXISTING_CONFERENCE	このエラーは、会議が存在していないことを示します。
static int	CTIERR_NO_RECORDING_SESSION	このエラーは、録音セッションがないために録音要求に失敗したことを示します。
static int	CTIERR_NO_RESPONSE_FROM_MP	このエラーは、メディア リソースからの応答がないことを示します。
static int	CTIERR_NOT_PRESERVED_CALL	このエラーは、コールが維持されていないことを示します。
static int	CTIERR_OPERATION_FAILED_QUIETCLEAR	このエラーは、一時的な障害が原因で、このコールの機能が利用できないことを示します。
static int	CTIERR_OPERATION_NOT_ALLOWED	このエラーは、この操作が許可されていないことを示します。
static int	CTIERR_OUT_OF_BANDWIDTH	このエラーは、帯域幅の範囲外であることを示します。
static int	CTIERR_OWNER_NOT_ALIVE	このエラーは、デバイスの登録が失敗したことを示します。
static int	CTIERR_PENDING_ACCEPT_OR_ANSWER_REQUEST	このエラーは、保留中の受け入れ要求または応答要求があることを示します。
static int	CTIERR_PENDING_START_MONITORING_REQUEST	このエラーは、保留中のモニタリング開始要求があることを示します。
static int	CTIERR_PENDING_START_RECORDING_REQUEST	このエラーは、保留中の録音開始要求があることを示します。
static int	CTIERR_PENDING_STOP_RECORDING_REQUEST	このエラーは、保留中の録音停止要求があることを示します。
static int	CTIERR_PRIMARY_CALL_INVALID	このエラーは、モニタリング要求のプライマリ コールが無効であるか、またはアイドル状態であることを示します。
static int	CTIERR_PRIMARY_CALL_STATE_INVALID	このエラーは、モニタリング要求のプライマリ コールが無効な状態であることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	CTIERR_RECORDING_ALREADY_INPROGRESS	このエラーは、録音がすでに進行中であるために録音要求が失敗したことを示します。
static int	CTIERR_RECORDING_CONFIG_NOT_MATCHING	このエラーは、録音設定が一致しないことを示します。
static int	CTIERR_RECORDING_SESSION_INACTIVE	このエラーは、録音セッションが非アクティブであるために録音要求に失敗したことを示します。
static int	CTIERR_REDIRECT_UNAUTHORIZED_COMMAND_USAGE	このエラーは、未承認のコマンド使用のリダイレクトを示します。
static int	CTIERR_REGISTER_FEATURE_ACTIVATION_FAILED	このエラーは、登録機能のアクティベーションに失敗したことを示します。
static int	CTIERR_REGISTER_FEATURE_APP_ALREADY_REGISTERED	機能登録アプリケーションがすでに登録されています。
static int	CTIERR_REGISTER_FEATURE_PROVIDER_NOT_REGISTERED	機能登録プロバイダーが登録されていません。
static int	CTIERR_RESOURCE_NOT_AVAILABLE	このエラーは、要求を処理するためにリソースを利用できないことを示します。
static int	CTIERR_START_MONITORING_FAILED	このエラーは、モニタリング開始要求に失敗したことを示します。
static int	CTIERR_START_RECORDING_FAILED	このエラーは、録音開始要求に失敗したことを示します。
static int	CTIERR_STATION_SHUT_DOWN	このエラーは、ステーションのシャットダウンが存在することを示します。
static int	CTIERR_SYSTEM_ERROR	このエラーは、CTI システム エラーを示します。
static int	CTIERR_UDP_PASS_THROUGH_NOT_SUPPORTED	このエラーは、UDP データ パススルーがサポートされないことを示します。
static int	CTIERR_UNKNOWN_EXCEPTION	このエラーは、不明な例外が発生したことを示します。
static int	CTIERR_UNSUPPORTED_CALL_PARK_TYPE	このエラーは、コール パークのタイプがサポートされないことを示します。
static int	CTIERR_UNSUPPORTED_CFWD_TYPE	このエラーは、コール転送タイプがサポートされないことを示します。
static int	CTIERR_USER_NOT_AUTH_FOR_SECURITY	このエラーは、ユーザがセキュア接続のために認証されていないことを示します。
static int	CTIERR_REDIRECT_CALL_PARTITIONING_POLICY	このエラーは、リダイレクトが認証されていないことを示します。
static int	CTIERR_FEATURE_NOT_AVAILABLE	このエラーは、機能が利用できないことを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	DARES_INVALID_REQ_TYPE	このエラーは、コール処理内部エラー、DaRes 無効要求タイプが存在することを示します。
static int	DATA_SIZE_LIMIT_EXCEEDED	このエラーは、XML データ オブジェクト サイズが、許容値よりも大きいことを示します。
static int	DB_ERROR	このエラーは、デバイス クエリーに不正なデバイス タイプがあることを示します。
static int	DB_ILLEGAL_DEVICE_TYPE	このエラーは、DB に不正なデバイス タイプがあることを示します。
static int	DB_NO_MORE_DEVICES	このエラーは使用されなくなりました。
static int	DESTINATION_BUSY	このエラーは、転送先が通話中であることを示します。
static int	DESTINATION_UNKNOWN	このエラーは、転送先が見つからないことを示します。
static int	DEVICE_ALREADY_REGISTERED	このエラーは、デバイスがすでに登録されているためにデバイス登録が失敗したことを示します。
static int	DEVICE_NOT_OPEN	このエラーは、デバイスがオープンされていないか、または登録されていないため、回線のオープンに失敗したことを示します。
static int	DEVICE_OUT_OF_SERVICE	このエラーは、デバイスがアウトオブサービスであることを示します。
static int	DIGIT_GENERATION_ALREADY_IN_PROGRESS	このエラーは、番号の生成が進行中であることを示します。
static int	DIGIT_GENERATION_CALLSTATE_CHANGED	このエラーは、コールの状態が無効で継続できないことを示します。
static int	DIGIT_GENERATION_WRONG_CALL_HANDLE	このエラーは、コール ハンドルが無効で、コールが存在しない可能性があることを示します。
static int	DIGIT_GENERATION_WRONG_CALL_STATE	このエラーは、コールの状態が無効で、番号を生成できないことを示します。
static int	DIRECTORY_LOGIN_FAILED	このエラーは、ディレクトリ ログインに失敗し、ディレクトリが初期化されないことを示します。
static int	DIRECTORY_LOGIN_NOT_ALLOWED	このエラーは、ディレクトリ ログインに失敗したことを示します。
static int	DIRECTORY_TEMPORARY_UNAVAILABLE	このエラーは、ディレクトリが一時的に利用不能になっていることを示します。
static int	EXISTING_FIRSTPARTY	このエラーは、すでにデバイス制御メディアが存在していることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	HOLDFAILED	このエラーは、保留が回線制御層またはコール制御層によって拒否されたことを示します。
static int	ILLEGAL_CALLINGPARTY	このエラーは、デバイス上にない発側を使用して、発呼しようとしたことを示します。
static int	ILLEGAL_CALLSTATE	このエラーは、要求を呼び出すには、回線が正しい状態にないことを示します。
static int	ILLEGAL_HANDLE	このエラーは、ハンドルが無効であることを示します。
static int	ILLEGAL_MESSAGE_FORMAT	このエラーは、QBE プロトコル エラーが存在することを示します。
static int	INCOMPATIBLE_PROTOCOL_VERSION	このエラーは、JTAPI と CTI のバージョンに互換性がないことを示します。CTI エラー プロトコルのバージョンをサポートしません。
static int	INVALID_LINE_HANDLE	このエラーは、無効な回線ハンドルで回線操作を実行しようとしたことを示します。
static int	INVALID_RING_OPTION	このエラーは、呼出音オプションが無効であることを示します。
static int	LINE_GREATER_THAN_MAX_LINE	このエラーは、回線がこのデバイス上で利用できる最大回線数よりも多いことを示します。
static int	LINE_INFO_DOES_NOT_EXIST	このエラーは、回線情報がデータベースに存在しないことを示します。
static int	LINE_NOT_PRIMARY	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	LINECONTROL_FAILURE	このエラーは、新規のコールの状態が原因で、回線制御がそのコールの開始を拒否したことを示します。
static int	MAX_NUMBER_OF_CTI_CONNECTIONS_REACHED	CTI 接続の最大数に達しました。
static int	MSGWAITING_DESTN_INVALID	このエラーは、無効な DN に対してメッセージ待ちランプをセットしようとしたことを示します。メッセージ待ち宛先が見つかりません。
static int	NO_ACTIVE_DEVICE_FOR_THIRDPARTY	このエラーは、サード パーティに対してアクティブなデバイスがないことを示します。
static int	NO_CONFERENCE_BRIDGE	このエラーは、会議ブリッジがないことを示します。
static int	NOT_INITIALIZED	このエラーは、CTI の初期化が完了する前に、プロバイダーをオープンしようとしたことを示します。
static int	PROTOCOL_TIMEOUT	コール制御から内部エラーが返されました。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	PROVIDER_ALREADY_OPEN	このエラーは、プロバイダーを再オープンしようとしたことを示します。
static int	PROVIDER_CLOSED	すでにクローズされているプロバイダーをクローズしようとした。
static int	PROVIDER_NOT_OPEN	このエラーは、デバイスの一覧が不完全であるか、デバイスの一覧の照会がタイムアウトになったか、または照会が中断されたことを示します。
static int	REDIRECT_CALL_CALL_TABLE_FULL	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_CALL_DESTINATION_BUSY	このエラーは、転送先が通話中であることを示します。
static int	REDIRECT_CALL_DESTINATION_OUT_OF_ORDER	このエラーは、リダイレクト先が故障中ことを示します。
static int	REDIRECT_CALL_DIGIT_ANALYSIS_TIMEOUT	このエラーは、ディジット分析がタイムアウトになったことを示します。これはコール制御から返された内部エラーです。
static int	REDIRECT_CALL_DOES_NOT_EXIST	このエラーは、存在しないまたは現在有効ではないコールをリダイレクトしようとしたことを示します。
static int	REDIRECT_CALL_INCOMPATIBLE_STATE	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_CALL_MEDIA_CONNECTION_FAILED	このエラーは、メディア接続エラーを示します。これはコール制御から返された内部エラーです。
static int	REDIRECT_CALL_NORMAL_CLEARING	このエラーは、通常のコールのクリアのためにリダイレクトが失敗したことを示します。
static int	REDIRECT_CALL_ORIGINATOR_ABANDONED	このエラーは、リダイレクトされるコールの遠端がハングアップしていることを示します。
static int	REDIRECT_CALL_PARTY_TABLE_FULL	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_CALL_PENDING_REDIRECT_TRANSACTION	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_CALL_PROTOCOL_ERROR	このエラーは、プロトコル エラーを示します。これはコール制御から返された内部エラーです。
static int	REDIRECT_CALL_UNKNOWN_DESTINATION	このエラーは、不明な宛先にリダイレクトしようとしたことを示します。
static int	REDIRECT_CALL_UNKNOWN_ERROR	このエラーは、コール制御から内部エラーが返されたことを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	REDIRECT_CALL_UNKNOWN_PARTY	このエラーは、不明な通話者が検出されたことを示します。これはコール制御から返された内部エラーです。
static int	REDIRECT_CALL_UNRECOGNIZED_MANAGER	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_CALLINFO_ERR	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	REDIRECT_ERR	このエラーは、コール制御から内部エラーが返されたことを示します。
static int	RETRIEVEFAILED	このエラーは、コールの取得が回線制御またはコール制御によって拒否されたことを示します。
static int	RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	このエラーは、回線に他のコールがすでに存在するために、保留コールの取得中にエラーが発生したことを示します。
static int	SSAPI_NOT_REGISTERED	このエラーは、内部サポート インターフェイスが初期化されていないときに、リダイレクト コマンドが発行されたことを示します。CTI が初期化を完了していないか、内部エラーが生じています。
static int	TIMEOUT	このエラーは、要求がタイムアウトになったことを示します。
static int	TRANSFER_INACTIVE	このエラーは、コンサルト転送が存在しないときに、転送を実行しようとしたことを示します。
static int	TRANSFERFAILED	このエラーは、コール レッグの 1 つが遠端でハング アップしているか、接続解除されたために、転送が失敗した可能性があることを示します。
static int	TRANSFERFAILED_CALLCONTROL_TIMEOUT	このエラーは、転送の間、コール制御から予期される応答を受信していないことを示します。
static int	TRANSFERFAILED_DESTINATION_BUSY	このエラーは、使用中状態の宛先に転送しようとしたことを示します。
static int	TRANSFERFAILED_DESTINATION_UNALLOCATED	このエラーは、登録されていない電話番号に転送しようとしたことを示します。
static int	TRANSFERFAILED_OUTSTANDING_TRANSFER	このエラーは、既存の転送が進行中であることを示します。
static int	UNDEFINED_LINE	このエラーは、指定された回線が、デバイスで見つからないことを示します。
static int	UNKNOWN_GLOBAL_CALL_HANDLE	このエラーは、グローバル コール ハンドルが不明であることを示します。

表 6-82　CiscoJtapiException のフィールド （続き）

インターフェイス	フィールド	説明
static int	UNRECOGNIZABLE_PDU	このエラーは、QBE プロトコル エラーが存在することを示します。
static int	UNSPECIFIED	このエラーは、詳細不明のエラーが発生したことを示します。

継承したフィールド

なし

メソッド

表 6-83　CiscoJtapiException のメソッド

インターフェイス	メソッド	説明
int	getErrorCode()	この例外の errorCode を整数として返します。
java.lang.String	getErrorDescription()	errorCode の詳細な説明を返します。
java.lang.String	getErrorDescription(int errorCode)	**推奨されません。** 代わりに、String getErrorDescription (); を使用してください。errorCode の詳細な説明を返します。
java.lang.String	getErrorName()	文字列の形式で例外を返します。
java.lang.String	getErrorName(int errorCode)	**推奨されません。** 代わりに、String getErrorName (); を使用してください。文字列の形式で例外を返します。

継承したメソッド

なし

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoJtapiPeer

CiscoJtapiPeer は、com.cisco.services.tracing.TraceModule インターフェイスを拡張することによって、アプリケーションからトレース情報へのアクセスを可能にします。Cisco JTAPI の実装によって作成された JtapiPeer オブジェクトのインスタンスはすべて、このインターフェイスを実装します。Cisco JTAPI 実装のトレース設定を操作する必要があるアプリケーションは、CiscoJtapiPeer.getTraceManager メソッドを使用して TraceManager オブジェクトを取得できます。アプリケーションは TraceManager オブジェクトを com.cisco.services.tracing パッケージに記述されたとおりに操作できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoObjectContainer, javax.telephony.JtapiPeer, TraceModule

宣言

public interface CiscoJtapiPeer extends TraceModule, javax.telephony.JtapiPeer, CiscoObjectContainer

フィールド

なし

メソッド

表 6-84　CiscoJtapiPeer のメソッド

インターフェイス	メソッド	説明
CiscoJtapiProperties	getJtapiProperties()	アプリケーションが使用できる各種のメソッドを定義します。アプリケーションはそれらのメソッドによって、JTAPI レイヤで使用されるパラメータを変更できます。

表 6-84　CiscoJtapiPeer のメソッド （続き）

インターフェイス	メソッド	説明
void	setJtapiProperties(CiscoJtapiProperties jtapiproperties)	jtapi.ini ファイルの CiscoJtapiProperties で行われた変更を保存する機能をアプリケーションに付与し、JTAPIPeer のプロバイダーのプロパティでこれらの変更を有効にします。

継承したメソッド

インターフェイス com.cisco.services.tracing.TraceModule から
getTraceManager, getTraceModuleName

インターフェイス javax.telephony.JtapiPeer から
getName, getProvider, getServices

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

詳細については、CiscoJtapiProperties と TraceModule を参照してください。

CiscoJtapiProperties

Cisco Unified JTAPI の動作と機能は多くのパラメータによって調整されます。それらのパラメータは、CiscoJtapiPeer がインスタンス化される際に、jtapi.ini ファイルから読み込まれます。アプリケーションは、この CiscoJtapiproperties インターフェイスを介して、これらのパラメータを制御できます。

アプリケーションは CiscoJtapiproperties プロパティ オブジェクトを照会して、そのアプリケーションの機能に適するようにこれらのパラメータを変更できます。また、CiscoJtapiproperties インターフェイスを介してプロパティにアクセスすることにより、（アプリケーション側から）これらのパラメータを管理する場所が一元化されます。最も可視性の高いパラメータは、トレース レベルとトレース対象を定義しているパラメータです。

使用法

```
JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
 if(peer instanceof CiscoJtapiPeer) {
 CiscoJtapiProperties jProps = ((CiscoJtapiPeer)peer).getJtapiProperties();
 jProps.setTracePath("¥¥D:¥¥Traces¥¥WorkFlow"); jProps.setUseJavaConsoleTrace(false);
 MyProviderObserver providerObserver = new MyProviderObserver ();
 provider = peer.getProvider ( providerName ); }
```

上記の例では、アプリケーションで Java コンソールのトレースをオフに設定し、トレースのパスを D:¥Traces¥WorkFlowApp1 に設定しています。ピアが取得されると、jtapi.ini ファイル内で設定されているパラメータが読み込まれ、CiscoJtapiProperties を実装するオブジェクトが作成されます。

jtapi.ini ファイルがクラスパスに存在しない場合は、デフォルト設定を使用してこのオブジェクトが作成されます。getProvider () の最初の呼び出し時に、Cisco Jtapi で使用されるパラメータが読み込まれて確定されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoJtapiProperties

フィールド

なし

メソッド

表 6-85　CiscoJtapiProperties のメソッド

インターフェイス	メソッド	説明
void	deleteCertificates(java.lang.String username, java.lang.String instanceID, java.lang.String ccmCAPFAddress, java.lang.String certificatePath)	証明書ストアにインストールされた USER インスタンスの X.509 クライアント証明書を削除します。
void	deleteSecurityPropertyForInstance(java.lang.String username, java.lang.String instanceID, java.lang.String capfIp, java.lang.String certPath)	jtapi.ini ファイルからセキュリティを適切に削除し、ユーザ名またはインスタンス ID に対して以前にインストールされた証明書も削除します。
java.lang.String	getAlarmServiceHostname()	アラーム サービスのホスト名を取得します。
int	getAlarmServicePort()	アラーム サービスのポート番号を取得します。
boolean	getCallSecurityStatusChangedEv()	イベント CallSecurityStatusChangedEv を受け取るかどうかをアプリケーションに指示します（該当する場合）。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
int	getCtiRequestTimeout()	プロバイダーのオープンを除く CTI 要求のタイムアウトを取得します（秒）。
java.lang.String[]	getDebuggingNames()	サポートされている jtapi デバッグ レベル トレースの名前を取得します。
boolean	getDebuggingValue(java.lang.String debuggingName)	デバッグ レベル トレースの有効状態または無効状態を取得します。
int	getDesiredServerHeartbeatInterval()	CTI Manager が JTAPI にハートビートを送信する間隔として指定されている時間を取得します（秒）。
boolean	getDiscConnBeforeCreatingInCPIC()	リダイレクト先だけがアプリケーションによって監視される場合のシナリオのイベントの順序を制御します。このインターフェイスは、ConnDisconnectedEv が ConnCreatedEv の前に送信される場合に true を返し、そうでない場合は false を返します。
java.lang.String	getFileNameBase()	各ログ ファイルのファイル名。
java.lang.String	getFileNameExtension()	ログ ファイルのファイル名拡張子を取得します。
int	getJavaSocketConnectTimeout()	SOCKET CONNECT TIMEOUT のサービス パラメータ値を返します（秒）。
int	getNumTraceFiles()	ロールオーバー前のトレース ファイルの数を取得します。
boolean	getPeriodicWakeupEnabled()	定期起動の有効状態を取得します。
int	getPeriodicWakeupInterval()	定期起動の間隔を取得します（ミリ秒）。
boolean	getProcessOfferingAfterNewcallevent()	jtapi.ini パラメータ ProcessOfferringAfterNewcallEvent' のブール値を取得します。デフォルトでは、このインターフェイスは false を返します。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
int	getProviderOpenRequestTimeout()	プロバイダー オープン要求のタイムアウトを設定します（秒）。
int	getProviderOpenRetryAttempts()	CTI Manager への再接続試行の最大数のサービス パラメータ値を返します。
int	getProviderRetryInterval()	CTI Manager への接続再試行の間隔を取得します（秒）。
int	getQueueSizeThreshold()	アラームをトリガするイベント キュー サイズのしきい値を取得します。
boolean	getQueueStatsEnabled()	イベント キュー統計値の有効状態を取得します。
int	getRouteSelectTimeout()	ルート選択のタイムアウトを取得します（ミリ秒）。
java.util.Hashtable	getSecurityPropertyForInstance()	ユーザおよび InstanceIDs のすべてのパラメータが設定されたハッシュ テーブルを返します。 キーと値のペアについては、「User/InstanceID のハッシュ テーブル」（P.6-172）を参照してください。
java.util.Hashtable	getSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID)	ユーザおよび InstanceIDs のすべてのパラメータが設定されたハッシュ テーブルを返します。 キーと値のペアについては、「User/InstanceID のハッシュ テーブル」（P.6-172）を参照してください。
java.lang.String[]	getServices()	この実装がサポートしているサービスを返します。
java.lang.String	getSyslogCollector()	syslog コレクタのホスト名を取得します。
int	getSyslogCollectorUDPPort()	syslog コレクタの UDP ポートを取得します。
java.lang.String	getTraceDirectory()	トレース ファイルの書き込み先のパス ディレクトリ。
int	getTraceFileSize()	ロールオーバー前のトレース ファイルのサイズ。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
java.lang.String[]	getTraceNames()	サポートされている jtapi トレースの名前を取得します。
java.lang.String	getTracePath()	トレース ファイルが配置されるパスを取得します。
boolean	getTraceValue(java.lang.String traceName)	トレースの有効状態または無効状態を取得します。
boolean	getUpdateJtapiCalledWithOriginalCalled()	更新された着側アドレスに対する Jtapi の動作を変更するパラメータ設定を照会します。
boolean	getUseAlarmService()	アラーム サービスの有効状態または無効状態を取得します。
boolean	getUseFileTrace()	jtapi ログ ファイル トレースの有効状態または無効状態を取得します。
boolean	getUseJavaConsoleTrace()	jtapi コンソール トレースの有効状態または無効状態を取得します。
boolean	getUseSameDir()	UseSameDir が true の場合は、同じディレクトリにトレース ファイルが書き込まれます。
boolean	getUseSyslog()	syslog トレースの有効状態または無効状態を取得します。
boolean	IsCertificateUpdated(java.lang.String user, java.lang.String instanceID)	指定したユーザ/インスタンス ID のクライアントおよびサーバの証明書が更新される場合、またはクライアントおよびサーバの証明書が更新されていない場合に関する情報を提供します。
void	setAlarmServiceHostname(java.lang.String hostname)	アラーム サービスのホスト名を設定します。
void	setAlarmServicePort(int portNumber)	アラーム サービスの受信を行うポート番号を設定します。
void	setCallSecurityStatusChangedEv(boolean val)	アプリケーションで、CallSecurityStatusChangedEv が true または false のどちらかを受信するためのフィルタを設定できます。

表 6-85　CiscoJtapiProperties のメソッド（続き）

インターフェイス	メソッド	説明
void	setCtiRequestTimeout(int seconds)	プロバイダーのオープンを除く CTI 要求のタイムアウトを設定します（秒）。
void	setDebuggingValue(java.lang.String debuggingName, boolean value)	特定のデバッグ レベル トレースを有効または無効にします。
void	setDesiredServerHeartbeatInterval(int seconds)	CTI Manager が JTAPI にハートビートを送信する間隔として指定されている時間を設定します（秒）。
void	setDiscConnBeforeCreatingInCPIC(boolean val)	イベントの順序を設定し、リダイレクト先でのリダイレクト中に作成される Connection よりも前に、Disconnect を送信します。
void	setFileNameBase(java.lang.String base)	ログ ファイルのファイル名を設定します。
void	setFileNameExtension(java.lang.String extn)	ログ ファイルのファイル名拡張子を設定します。
void	setJavaSocketConnectTimeout(int timeout)	アプリケーションでソケット接続タイムアウトを設定できるようにします（秒）。
void	setNumTraceFiles(int val)	ロールオーバー前のトレース ファイルの数を設定します。
void	setPeriodicWakeupEnabled(boolean enabled)	定期起動を有効または無効状態に設定します。
void	setPeriodicWakeupInterval(int milliseconds)	定期起動の間隔を設定します（ミリ秒）。
void	setProcessOfferingAfterNewcallevent(boolean val)	転送先だけがアプリケーションによって監視され、転送が Offering 状態で完了する転送シナリオのためのイベントの順序を制御します。
void	setProviderOpenRequestTimeout(int seconds)	プロバイダー オープン要求のタイムアウトを設定します（秒）。
void	setProviderOpenRetryAttempts(int retryAttempts)	アプリケーションで JTAPI の CTI Manager への再接続試行を設定できるようにします。
void	setProviderRetryInterval(int seconds)	CTI Manager への接続再試行の間隔を設定します（秒）。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
void	setQueueSizeThreshold(int size)	アラームをトリガするイベント キュー サイズのしきい値を設定します。
void	setQueueStatsEnabled(boolean enabled)	イベント キュー統計値を有効または無効にします。
void	setRouteSelectTimeout(int milliseconds)	ルート選択のタイムアウトを設定します（ミリ秒）。
void	setSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID, java.lang.String authCode, java.lang.String tftp, java.lang.String tftpPort, java.lang.String capf, java.lang.String capfPort, java.lang.String certPath, boolean securityOption)	**推奨されません。** このメソッドは、オーバーロードされたメソッド setSecurityPropertyForInstance に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。
void	setSecurityPropertyForInstance(java.lang.String user, java.lang.String instanceID, java.lang.String authCode, java.lang.String tftp, java.lang.String tftpPort, java.lang.String capf, java.lang.String capfPort, java.lang.String certPath, boolean securityOption, java.lang.String certstorePassphrase)	アプリケーションにサーバ/クライアント証明書をダウンロードする機能を提供し、JTAPI の jtapi.ini ファイル内のアプリケーション インスタンスのセキュリティ プロパティを設定します。
void	setServices(java.lang.String[] services)	利用可能なサービスのリストを設定します。
void	setSyslogCollector(java.lang.String value)	syslog コレクタのホスト名を設定します。
void	setSyslogCollectorUDPPort(int port)	syslog コレクタの UDP ポートを設定します。
void	setTraceDirectory(java.lang.String dir)	jtapi トレース ファイルの書き込み先のディレクトリを設定します。
void	setTraceFileSize(int val)	トレース ファイルのサイズを設定します。
void	setTracePath(java.lang.String path)	jtapi トレースの書き込み先のディレクトリ ルートを設定します。
void	setTraceValue(java.lang.String traceName, boolean value)	特定のトレースを有効または無効にします。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
void	setUpdateJtapiCalledWithOriginalCalled(boolean val)	このパラメータが True に設定されると、Jtapi 着信者情報が常に元の着信者で更新されます。
void	setUseAlarmService(boolean value)	アラーム サービスを有効または無効にします。
void	setUseFileTrace(boolean value)	jtapi ログ ファイル トレースを有効または無効にします。
void	setUseJavaConsoleTrace(boolean value)	jtapi コンソール トレースを有効または無効にします。
void	setUseSameDir(boolean value)	UseSameDir が true の場合は、同じディレクトリにトレース ファイルが書き込まれます。
void	setUseSyslog(boolean value)	syslog トレースを有効または無効にします。
void	updateCertificate(java.lang.String user, java.lang.String intanceID, java.lang.String authcode, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath)	**推奨されません。** このメソッドは、オーバーロードされたメソッド updateCertifcate に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。
void	updateCertificate(java.lang.String user, java.lang.String intanceID, java.lang.String authcode, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath, java.lang.String certStorePassphrase)	証明書ストアに USER インスタンスの X.509 クライアント証明書をインストールします。
void	updateServerCertificate(java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath)	**推奨されません。** このメソッドは、オーバーロードされたメソッド updateServerCertifcate に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。

表 6-85　CiscoJtapiProperties のメソッド （続き）

インターフェイス	メソッド	説明
void	updateServerCertificate(java.lang.String userName, java.lang.String instanceID, java.lang.String ccmTFTPAddress, java.lang.String ccmTFTPPort, java.lang.String ccmCAPFAddress, java.lang.String ccmCAPFPort, java.lang.String certificatePath, java.lang.String certStorePassphrase)	指定した証明書パスに X.509 サーバ証明書をインストールします。

User/InstanceID のハッシュテーブル

表 6-86　User/InstanceID のハッシュテーブル

キー	値
「user」	userName
文字列「instanceID」	InstanceID
文字列「Authcode」	authCode
文字列「CAPF」	capfServerIP-Address
文字列「CAPFPort」	capfServer の IP アドレス / ポート
文字列「TFTP」	tftpServer の IP アドレス
文字列「TFTPPort」	tftpServer の IP アドレス / ポート
文字列「CertPath」	証明書パス
文字列「securityOption」	セキュリティ オプションを表すブール値（有効なら true、無効なら false）。
文字列「certificateStatus」	証明書のステータスを表すブール値（更新済みなら true、未更新なら false）。

関連資料

CiscoLocales

このインターフェイスは、Cisco Unified JTAPI がサポートするすべてのロケールを列挙します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoLocales

フィールド

表 6-87　CiscoLocales のフィールド

インターフェイス	フィールド
static int	LOCALE_ARABIC_ALGERIA
static int	LOCALE_ARABIC_BAHRAIN
static int	LOCALE_ARABIC_EGYPT
static int	LOCALE_ARABIC_IRAQ
static int	LOCALE_ARABIC_JORDAN
static int	LOCALE_ARABIC_KUWAIT
static int	LOCALE_ARABIC_LEBANON
static int	LOCALE_ARABIC_MOROCCO
static int	LOCALE_ARABIC_OMAN
static int	LOCALE_ARABIC_QATAR
static int	LOCALE_ARABIC_SAUDI_ARABIA
static int	LOCALE_ARABIC_TUNISIA
static int	LOCALE_ARABIC_UNITED_ARAB_EMIRATES
static int	LOCALE_ARABIC_YEMEN
static int	LOCALE_BULGARIAN_BULGARIA
static int	LOCALE_CATALAN_SPAIN
static int	LOCALE_CHINESE_HONG_KONG
static int	LOCALE_CROATIAN_CROATIA

表 6-87　CiscoLocales のフィールド

インターフェイス	フィールド
static int	LOCALE_CZECH_CZECH_REPUBLIC
static int	LOCALE_DANISH_DENMARK
static int	LOCALE_DUTCH_NETHERLAND
static int	LOCALE_ENGLISH_UNITED_KINGDOM
static int	LOCALE_ENGLISH_UNITED_STATES
static int	LOCALE_FINNISH_FINLAND
static int	LOCALE_FRENCH_FRANCE
static int	LOCALE_GERMAN_GERMANY
static int	LOCALE_GREEK_GREECE
static int	LOCALE_HEBREW_ISRAEL
static int	LOCALE_HUNGARIAN_HUNGARY
static int	LOCALE_ITALIAN_ITALY
static int	LOCALE_JAPANESE_JAPAN
static int	LOCALE_KOREAN_KOREA
static int	LOCALE_NORWEGIAN_NORWAY
static int	LOCALE_POLISH_POLAND
static int	LOCALE_PORTUGUESE_BRAZIL
static int	LOCALE_PORTUGUESE_PORTUGAL
static int	LOCALE_ROMANIAN_ROMANIA
static int	LOCALE_RUSSIAN_RUSSIA
static int	LOCALE_SERBIAN_REPUBLIC_OF_MONTENEGRO
static int	LOCALE_SERBIAN_REPUBLIC_OF_SERBIA
static int	LOCALE_SIMPLIFIED_CHINESE_CHINA
static int	LOCALE_SLOVAK_SLOVAKIA
static int	LOCALE_SLOVENIAN_SLOVENIA
static int	LOCALE_SPANISH_SPAIN
static int	LOCALE_SWEDISH_SWEDEN

表 6-87　CiscoLocales のフィールド

インターフェイス	フィールド
static int	LOCALE_THAI_THAILAND
static int	LOCALE_TRADITIONAL_CHINESE_CHINA

メソッド

なし

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoMediaConnectionMode

CiscoMediaConnectionMode インターフェイスは、すべてのメディア接続モードを列挙します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoMediaConnectionMode

フィールド

表 6-88　CiscoMediaConnectionMode のフィールド

インターフェイス	フィールド	説明
static int	NONE	送信チャネルも受信チャネルもアクティブではありません。
static int	RECEIVE_ONLY	受信チャネルだけがアクティブです。
static int	TRANSMIT_AND_RECEIVE	送信チャネルも受信チャネルもアクティブです。
static int	TRANSMIT_ONLY	送信チャネルだけがアクティブです。

メソッド

なし

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoMediaEncryptionAlgorithmType

CiscoMediaEncryptionAlgorithmType インターフェイスは、暗号化に使用される SRTP アルゴリズム タイプを示します。このインターフェイスは、アプリケーションが iscoRTPInputKeyEv および CiscoRTPOutputKeyEv で取得できるすべてのセキュリティ インジケータ値を列挙します。アプリケーションが CTIPort およびメディアの終端 RP 上で独自のメディアを終了させている場合、次のアルゴリズムのうち 1 つだけを登録 API で提供する必要があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース	説明
3.x	拡張が追加されました。

スーパーインターフェイス

public interface CiscoMediaEncryptionAlgorithmType

フィールド

表 6-89　CiscoMediaEncryptionAlgorithmType のフィールド

インターフェイス	フィールド	説明
static int	AES_128_COUNTER	使用されるアルゴリズムは、Advanced Encryption Standard（AES; 高度暗号化規格）に基づきます。これは、コンピュータのセキュリティ標準です。暗号化スキームは、128 ビットのデータ ブロックを暗号化および復号化する対称ブロック サイファです。

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoMediaEncryptionKeyInfo

CiscoMediaEncryptionKeyInfo インターフェイスでは、アプリケーションが SRTP キーに関する情報を取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoMediaEncryptionKeyInfo

フィールド

なし

メソッド

表 6-90　CiscoMediaEncryptionKeyInfo のメソッド

インターフェイス	メソッド	説明
int	getAlgorithmID()	現在のストリームのメディア暗号化アルゴリズム ID を返します。
int	getIsMKIPresent()	MKI が存在するかどうかを示します。
byte[]	getKey()	ストリームのマスター鍵を返します。
int	getKeyLength()	このキーの keyLength を返します。
byte[]	getSalt()	ストリームのソルト鍵を返します。
int	getSaltLength()	このソルトの saltLength を返します。
int	keyDerivationRate()	このセッションの SRTP 鍵逸脱率を示します。

関連資料

CiscoRTPInputKeyEv と CiscoRTPOutputKeyEv を参照してください。

CiscoMediaOpenLogicalChannelEv

CiscoMediaOpenLogicalChannelEv イベントは、動的に登録される CiscoMediaTerminal または CiscoRouteTerminal に対してメディアが確立されるたびに送信されます。アプリケーションは、このイベントを受信すると、CiscoMediaTerminal または CiscoRouteTerminal に対して setRTPParams を起動し、メディアを終端する IP アドレスおよびポート番号をこのイベントで配信される rtpHandle とともに渡す必要があります。

アプリケーションは、CiscoProvider.getCall（CiscoRTPHandle）を使用してコール参照を取得できます。アプリケーションは、setRTPParams メソッドが起動されない限り、遠端やローカル エンドが機能を起動できないことに注意する必要があります。アプリケーションが指定時間内にこのイベントに応答しない場合、コールは切断されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.0(1)	getAddressingModeForMedia() メソッドが追加されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoMediaOpenLogicalChannelEv extends CiscoTermEv

フィールド

表 6-91　CiscoMediaOpenLogicalChannelEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-92　CiscoMediaOpenLogicalChannelEv のメソッド

インターフェイス	メソッド	説明
int	getAddressingModeForMedia()	int Application を返し、要求された IP アドレッシング モードに対する次の値を受け取ります。 • CiscoTerminal.IP_ADDRESSING_IPv4：アプリケーションが setRTPParams 要求で IPv4 形式の IP アドレスを提供する必要があることを意味します。 • CiscoTerminal.IP_ADDRESSING_IPv6：アプリケーションが set RTP Params 要求で IPv6 形式の IP アドレスを提供する必要があることを意味します。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、このハンドルを RTP パラメータとともに CiscoMediaTerminal または CiscoRouteTerminal に渡す必要があります。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。
int	getMediaConnectionMode()	CiscoMediaConnectionMode を返します。アプリケーションは次のいずれかの値を受け取ります。 • CiscoMediaConnectionMode.RECEIVE_ONLY：一方向メディアによる受信だけであることを意味します。 • CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE：双方向メディアであることを意味します。 通常、アプリケーションは値 NONE を受け取ることはありません。受け取った場合、アプリケーションはそのイベントを無視してエラーを記録します。

表 6-92　CiscoMediaOpenLogicalChannelEv のメソッド

インターフェイス	メソッド	説明
int	getPacketSize()	遠端のパケット サイズを返します（ミリ秒単位）。 getPacketSize
int	getPayLoadType()	次の定数の 1 つの遠端ペイロード形式を返します。 • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）および CiscoRTPParams を参照してください。

CiscoMediaSecurityIndicator

CiscoMediaSecurityIndicator は CiscoRTPInputKeyEv、CiscoRTPOutputKeyEv、および CiscoSnapShotRTPEv で送信されます。コール セキュリティ ステータスを表示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoMediaSecurityIndicator

フィールド

表 6-93　CiscoMediaSecurityIndicator のフィールド

インターフェイス	フィールド	説明
static int	MEDIA_ENCRYPT_KEYS_AVAILABLE	セキュリティを確保したモードでメディアを終端し、鍵を参照できます。
static int	MEDIA_ENCRYPT_KEYS_UNAVAILABLE	セキュリティを確保したモードでメディアを終端しますが、SRTP が Cisco Unified Communications Manager Administration で有効にされていないため、鍵は参照できません。
static int	MEDIA_ENCRYPT_USER_NOT_AUTHORIZED	セキュリティを確保したモードでメディアを終端しますが、ユーザが鍵を取得する権限を持っていないため、鍵は参照できません。
static int	MEDIA_NOT_ENCRYPTED	メディアのこのコールは暗号化されていません。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoMediaTerminal

CiscoMediaTerminal は、アプリケーションによる RTP メディア ストリームの終端を可能にする、特殊な CiscoTerminal です。CiscoMediaTerminal は、通常の CiscoTerminal と異なり、物理的なテレフォニー エンドポイントではないため、サードパーティ方式で監視、制御することができます。CiscoMediaTerminal は論理的なテレフォニー エンドポイントなので、メディアを終端する必要があるアプリケーションに関連付けることができます。そのようなアプリケーションには、音声メッセージング システム、Interactive Voice Response（IVR; 対話式音声自動応答）、ソフトフォンなどが含まれます。

（注）　Cisco Unified JTAPI により CiscoMediaTerminal として表現されるのは、CTIPort だけです。

メディアの終端プロセスには 2 つの段階があります。アプリケーションはまず、特定の端末向けのメディアを終端するために、Terminal.addObserver メソッドを使用して、CiscoTerminalObserver インターフェイスを実装するオブザーバを追加します。次に、CiscoMediaTerminal.register メソッドを使用して、その IP アドレス、およびその端末向けの着信 RTP ストリームの送信先となるポート番号を登録します。

コールごとに動的に IP アドレスおよびポート番号を提供するには、アプリケーションは、サポートしている機能だけを提示して登録する必要があります。アプリケーションは、メディアが確立されるたびに送信される CiscoMediaOpenLogicalChannelEv に応答する必要があります。このタイプに登録するアプリケーションは、メディアが確立されていない限り、このイベントが受信されたときに、遠端やローカル エンドがあらゆる機能を実行できないことに注意する必要があります。アプリケーションが指定時間内にこのイベントに応答しない場合、コールはドロップされます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1x	IPv6 のサポートが追加されました。

スーパーインターフェイス

CiscoObjectContainer, CiscoTerminal, javax.telephony.Terminal

宣言

public interface CiscoMediaTerminal extends CiscoTerminal

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoTerminal から
ASCII_ENCODING, DEVICESTATE_ACTIVE, DEVICESTATE_ALERTING, DEVICESTATE_HELD, DEVICESTATE_IDLE, DEVICESTATE_UNKNOWN, DEVICESTATE_WHISPER, DND_OPTION_CALL_REJECT, DND_OPTION_NONE, DND_OPTION_RINGER_OFF, IN_SERVICE, IP_ADDRESSING_MODE_IPV4, IP_ADDRESSING_MODE_IPV4_V6, IP_ADDRESSING_MODE_IPV6, IP_ADDRESSING_MODE_UNKNOWN, IP_ADDRESSING_MODE_UNKNOWN_ANATRED, NOT_APPLICABLE, OUT_OF_SERVICE, UCS2UNICODE_ENCODING, UNKNOWN_ENCODING

メソッド

表 6-94　Cisco*MediaTerminal* のメソッド

インターフェイス	メソッド	説明
void	register(java.net.InetAddress address, int port, CiscoMediaCapability[] capabilities)	このメソッドは MediaTerminal を登録し、MediaTerminal が登録されると、このメソッドは正常に終了します。 CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 このメソッドには 3 種類の引数があります。 • 最初の引数は、この端末の RTP メディア ストリームが終端されるインターネット アドレスを指定します。 • 2 番目の引数は RTP パケットが向けられる UDP ポートを指定します。 • 3 番目の引数は、アプリケーションがこの端末に対してサポートする RTP 符号化方式の種類を示します。 **パラメータ** • address : この端末の着信 IPv4 RTP ストリームが終端されるインターネット アドレスを指定します。 • port : RTP ストリームの受信に使用されるこの端末の UDP ポート。 • capabilities : アプリケーションがこの端末に対してサポートする RTP 符号化方式の種類のリスト。 **例外** CiscoRegistrationException

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
void	register(java.net.InetAddress address, int port)	**推奨されません。** アドレスとポートを指定して Terminal を登録します。デフォルトの符号化方式は G.711（64 KHz u-law）、パケット サイズは 30 ミリ秒です。 **パラメータ** • address：この端末の着信 IPv4 RTP ストリームのインターネット アドレス • port：RTP ストリームの受信に使用されるこの端末の UDP ポート **例外** CiscoRegistrationException
void	register(java.net.InetAddress address, int port,CiscoMediaCapability[] capabilities, int[] algorithmIDs)	このメソッドは MediaTerminal を登録します。CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 MediaTerminal が登録されると、このメソッドは正常に終了します。このメソッドでは、アプリケーションが CTIManager で TLS リンクを確立し、Cisco Unified Communications Manager Administration でユーザに対して SRTP Enabled フラグを有効にする必要があります。そうしないと、PrivilegeViolationException が送出されます。 **パラメータ** • address：この端末の着信 IPv4 RTP ストリームのインターネット アドレス • port：RTP ストリームの受信に使用されるこの端末の UDP ポート • capabilities：この端末でサポートされている RTP 符号化方式のリスト • AlgorithmIDs ：この CTIPort がサポートする SRTP アルゴリズム AlgorithmIDs は、CiscoMediaEncryptionAlgorithmType のいずれかにする必要があります。 **例外** CiscoRegistrationException javax.telephony.PrivilegeViolationException

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
void	register(java.net.InetAddress address, int port, CiscoMediaCapability[] capabilities, int[] algorithmIDs, java.net.InetAddress address_v6, int activeAddressingMode)	CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 このメソッドが成功した場合は、MediaTerminal が登録されます。activeAddressingMode はアプリケーション IP アドレッシング機能を示します。アプリケーションが activeAddressingMode を CiscoTerminal.IP_ADDRESSING_MODE_IPv4 に指定する場合、address も指定する必要があります。 アプリケーションが activeAddressingMode を CiscoTerminal.IP_ADDRESSING_MODE_IPv6 に指定する場合、address_v6 も指定する必要があります。 アプリケーションが activeAddressingMode を CiscoTerminal.IP_ADDRESSING_MODE_IPv4_6 に指定する場合、address および address_v6 も指定する必要があります。 **メソッドの引数** このメソッドには 4 種類の引数があります。 • 最初の引数は、この端末の RTP メディア ストリームが終端されるインターネット アドレスを指定します。 • 2 番目の引数は RTP パケットが向けられる UDP ポートを指定します。 • 3 番目の引数は、アプリケーションがこの端末に対してサポートする RTP 符号化方式の種類を示します。 • 4 番目の引数は、アプリケーションがサポートする SRTP アルゴリズムを示します。 このメソッドは、アプリケーションが CTIManager で TLS リンクを確立している場合と、アプリケーションがユーザに対して CM の管理ページで SRTP Enabled フラグを有効にしている場合にだけ使用できます。そうでない場合は、PrivilegeViolationException が送出されます。 **メソッドの事後条件** MediaTerminal が登録されると、このメソッドは正常に終了します。

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
		パラメータ • address：この端末の着信 IPv4 RTP ストリームのインターネット アドレス。アプリケーションのアドレッシング モードに応じて、null になる場合もあります。 • port : RTP ストリームの受信に使用されるこの端末の UDP ポート • capabilities：この端末でサポートされている RTP 符号化方式のリスト • AlgorithmIDs ：この CTIPort がサポートする SRTP アルゴリズム AlgorithmIDs は、CiscoMediaEncryptionAlgorithmType のいずれか 1 つだけになります。 • address_v6：この端末の着信 IPv6 RTP ストリームのインターネット アドレス。activeAddressingMode に応じて、null になる場合もあります。 • activeAddressingMode：アプリケーションがこの CiscoMediaTerminal を登録する IP アドレッシング モード 次のいずれかになります。 • CiscoTerminal.IP_ADDRESSING_MODE_IPv4 • CiscoTerminal.IP_ADDRESSING_MODE_IPv6 • CiscoTerminal.IP_ADDRESSING_MODE_IPv4z_v6（**7.0 以降**） **例外** CiscoRegistrationException, javax.telephony.PrivilegeViolationException

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
void	register(CiscoMediaCapability[] capabilities)	このメソッドは、指定された CiscoMediaCapabilities で MediaTerminal を登録します。このメソッドは、各コールに対して IP アドレスおよびポートを動的に指定する場合にだけ使用します。 このメソッドで登録するアプリケーションは、コールごとに CiscoMediaOpenLogicalChannelEv を受信するので、このオブジェクトの setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。 CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 **メソッドの引数** 引数はアプリケーションがこの Terminal に対してサポートする RTP 符号化方式の種類を示します。 **メソッドの事後条件** CiscoMediaTerminal が登録されると、このメソッドは正常に終了します。 **パラメータ** capabilities：この端末でサポートされている RTP 符号化方式のリスト **例外** CiscoRegistrationException

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
void	register(CiscoMediaCapability[] capabilities, int[] algorithmIDs)	このメソッドでは、指定された CiscoMediaCapabilities およびサポートされる SRTP アルゴリズムで MediaTerminal を登録します。 このメソッドは、各コールに対して IP アドレスおよびポートを動的に指定し、SRTP アルゴリズムも指定する場合にだけ使用します。 このメソッドで登録するアプリケーションは、コールごとに CiscoMediaOpenLogicalChannelEv を受信するので、このオブジェクトの setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。 この形式の register() には、アプリケーションがサポートする SRTP アルゴリズムを指定する 2 番目のパラメータも必要です。 このメソッドでは、アプリケーションが CTIManager で TLS リンクを確立し、Cisco Unified Communications Manager Administration でユーザに対して SRTP Enabled フラグを有効にする必要があります。そうしないと、PrivilegeViolationException が送出されます。 CiscoMediaTerminal が登録されると、このメソッドは正常に終了します。 CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 **パラメータ** • capabilities：この端末でサポートされている RTP 符号化方式のリスト • algorithmIDs：この端末でサポートされている RTP アルゴリズムのリスト AlgorithmIDs は、CiscoMediaEncryptionAlgorithmType のいずれかにする必要があります。 **例外** CiscoRegistrationException javax.telephony.PrivilegeViolationException

表 6-94　Cisco*MediaTerminal* のメソッド（続き）

インターフェイス	メソッド	説明
void	register(CiscoMediaCapability[] capabilities, int[] algorithmIDs, int activeAddressingMode)	CiscoMediaTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このメソッドが成功した場合は、MediaTerminal が登録されます。指定された CiscoMediaCapabilities およびサポートされる SRTP アルゴリズムで端末を登録します。また、これは、アプリケーションが各コールに対して動的に ipAddress およびポートを指定することを示します。 このメソッドを登録するアプリケーションは、各コールに対して CiscoMediaOpenLogicalChannelEv を受信し、このオブジェクトに対する setRTPParams メソッドを使用して ipAddress とポート番号を指定する必要があります。 2 番目の引数は、アプリケーションがサポートする SRTP アルゴリズムを示します。このメソッドは、アプリケーションが CTIManager で TLS リンクを確立している場合と、アプリケーションがユーザに対して Cisco Unified Communications Manager Administration で SRTP Enabled フラグを有効にしている場合にだけ使用できます。そうでない場合は、PrivilegeViolationException が送出されます。 **メソッドの引数** 引数は、アプリケーションがこの端末に対してサポートする RTP 符号化の種類と、アプリケーションまたは CTIManager の障害を示します。 **メソッドの事後条件** CiscoMediaTerminal が登録されると、このメソッドは正常に終了します。 **パラメータ** • capabilities：この端末でサポートされている RTP 符号化方式のリスト • algorithmIDs：この端末でサポートされている RTP アルゴリズムのリスト AlgorithmIDs は、CiscoMediaEncryptionAlgorithmType のいずれか 1 つだけになります。 • activeAddressingMode：アプリケーションがこの CiscoMediaTerminal を登録する IP アドレッシングモード activeAddressingMode は次のいずれかです。 – CiscoTerminal.IP_ADDRESSING_MODE_IPv4 – CiscoTerminal.IP_ADDRESSING_MODE_IPv6 – CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6 **例外** CiscoRegistrationException javax.telephony.PrivilegeViolationException

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
void	setRTPParams(CiscoRTPHandle rtpHandle, CiscoRTPParams rtpParams)	アプリケーションは、IP アドレスおよび RTP ポート番号を設定して、コールのメディア ストリームを動的に行う場合にだけこのメソッドを使用します。この場合、機能だけを指定して MediaTerminal または CiscoRouteTeminal を登録する必要があります。 アプリケーションでは、terminalObserver で CiscoCallOpenLogicalChannel を受信したときに、このメソッドを起動する必要があります。アプリケーションは、CiscoCallOpenLogicalChannelEv で受信する rtpHandle を渡す必要があります。アプリケーションは、CiscoProvider.getRTPHandle(rtpHandle) メソッドを呼び出して CiscoCall 参照を取得できます。 これは、コール オブザーバが端末に追加されていない場合、このイベントの送信時にコール オブザーバがなかった場合、またはこのハンドルに関連付けられたコールがない場合には、null を返します。 **パラメータ** • rtpHandle：CiscoMediaCallOpenLogicalChannelEv から取得します。 • rtpParams：CiscoRTPParams のタイプで、コールごとにメディア端末の動的 RTP アドレスとポート番号を指定するために使用されます。 **例外** javax.telephony.InvalidStateException javax.telephony.InvalidArgumentException javax.telephony.PrivilegeViolationException
void	unregister()	このメソッドは MediaTerminal を登録解除し、MediaTerminal が登録解除されると、このメソッドは正常に終了します。CiscoMediaTerminal は登録されている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。 **例外** CiscoUnregistrationException
boolean	isRegistered()	CiscoMediaTerminal が登録されている場合は true、そうでない場合は false を返します。MediaTerminal の場合、このメソッドは MediaTerminal が InService 状態の場合は true、OutOfService の場合は false を返します。CTIManager の障害の場合は、false を返します。

表 6-94　Cisco*MediaTerminal* のメソッド （続き）

インターフェイス	メソッド	説明
boolean	isRegisteredByThisApp()	このメソッドは、このアプリケーションが登録要求を発行して成功した場合に true を返します。この登録は、CTIManager の障害によりデバイスがアウトオブサービス状態にある場合でも有効です。これは、このアプリケーションがデバイスの登録を解除するまで true に設定されます。
int	getIPAddressingMode()	アプリケーションはこの API を起動して CiscoMediaTerminal の IP アドレッシング モードを照会できます。アドレッシング モードを表す定数は次のとおりです。 • CiscoTerminal.IP_ADDRESSING_IPv4 • CiscoTerminal.IP_ADDRESSING_IPv6 • CiscoTerminal.IP_ADDRESSING_IPv4_v6

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoTerminal から
createSnapshot, getAltScript, getDeviceState, getDNDOption, getDNDStatus, getEMLoginUsername, getFilter, getLocale, getProtocol, getRegistrationState, getRTPInputProperties, getRTPOutputProperties, getState, getSupportedEncoding, isRestricted, sendData, sendData, setDNDStatus, setFilter, unPark

インターフェイス javax.telephony.Terminal から
addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

CiscoTerminal と CiscoMediaOpenLogicalChannelEv.CiscoRTPParams を参照してください。

CiscoMonitorInitiatorInfo

このインターフェイスは、モニタリングの開始側に関する情報を定義します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoMonitorInitiatorInfo

フィールド

なし

メソッド

表 6-95　CiscoMonitorInitiatorInfo のメソッド

インターフェイス	メソッド	説明
CiscoAddress	getAddress()	モニタリングの開始側のアドレスを返します。
Int	getMonitorInitiatorCallLegHandle()	モニタリングの開始側のコール レッグのハンドルを返します。JTAPI では、provider.getCall (int monitorInitiatorCallLegHandle) を使用してモニタリング ターゲットのコールを取得します。 モニタリングの開始側のコールがこのプロバイダーでアクティブでない場合、このメソッドは null を返します。
java.lang.String	getTerminalName()	モニタリングの開始側の端末名を返します。

関連資料

なし

CiscoMonitorTargetInfo

このインターフェイスは、モニタリングのターゲットに関する情報を提供します。

宣言

public interface CiscoMonitorTargetInfo

フィールド

なし

メソッド

表 6-96　CiscoMonitorTargetInfo のメソッド

インターフェイス	メソッド	説明
CiscoAddress	getAddress()	モニタリングのターゲットのアドレスを返します。
Int	getMonitorTargetCallLegHandle()	モニタリングのターゲットのコール レッグのハンドルを返します。
java.lang.String	getTerminalName()	モニタリングのターゲットの端末名を返します。

関連資料

なし

CiscoObjectContainer

アプリケーションは ApplicationObject インターフェイスを使用して、アプリケーション定義オブジェクトを、このインターフェイスを実装するオブジェクトに関連付けることができます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

サブインターフェイス

CiscoAddress, CiscoCall, CiscoCallID, CiscoConnection, CiscoConnectionID, CiscoConsultCall, CiscoIntercomAddress, CiscoJtapiPeer, CiscoMediaTerminal, CiscoProvider, CiscoRouteTerminal, CiscoTerminal, CiscoTerminalConnection

宣言

public interface CiscoObjectContainer

フィールド

なし

メソッド

表 6-97　CiscoObjectContainer のメソッド

インターフェイス	メソッド	説明
java.lang.Object	getObject()	アプリケーション定義オブジェクトを取得します。
java.lang.Object	setObject(java.lang.Object reference)	アプリケーション定義オブジェクトを設定します。

関連資料

なし

CiscoOutOfServiceEv

CiscoOutOfServiceEv イベントは、アウトオブサービス イベント CiscoAddrOutOfServiceEv および CiscoTermOutOfServiceEv のスーパー クラスです。このクラスはアウトオブサービス イベントの原因を定義します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, javax.telephony.events.Ev

サブインターフェイス

CiscoAddrOutOfServiceEv または CiscoTermOutOfServiceEv

宣言

public interface CiscoOutOfServiceEv extends CiscoEv

フィールド

表 6-98　CiscoOutOfServiceEv のフィールド

インターフェイス	フィールド	説明
static int	CAUSE_CALLMANAGER_FAILURE	このイベントの原因は Cisco Unified Communications Manager の障害です。
static int	CAUSE_CTIMANAGER_FAILURE	このイベントの原因は CTIManager の障害です。
static int	CAUSE_DEVICE_FAILURE	このイベントの原因はデバイスの障害です。
static int	CAUSE_DEVICE_RESTRICTED	このイベントの原因はデバイスが制限されていることです。
static int	CAUSE_DEVICE_UNREGISTERED	このイベントの原因はデバイスが UNREGISTERED 状態であることです。
static int	CAUSE_LINE_RESTRICTED	このイベントの原因は回線が制限されていることです。
static int	CAUSE_NOCALLMANAGER_AVAILABLE	このイベントの原因は Cisco Unified Communications Manager の機能を利用できないことです。
static int	CAUSE_REHOME_TO_HIGHER_PRIORITY_CM	このイベントの原因は、ハイ プライオリティの Cisco Unified Communications Manager ノードへのフェールバックにおけるエラーです。
static int	CAUSE_REHOMING_FAILURE	このイベントの原因は rehome の試行中の障害です。
static int	ID	—

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

メソッド

なし

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoPartyInfo

このインターフェイスはコールの通話者情報を定義します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoPartyInfo

フィールド

表 6-99　CiscoPartyInfo のフィールド

インターフェイス	フィールド	説明
Static int	ABBREVIATED_NUMBER	この NumberType は 4 と同じで、発信者が Cisco Unified Communications Manager サーバと同じであることを示します。
Static int	INTERNATIONAL_NUMBER	この NumberType は 0 と同じで、何も設定されていないことを示します。
Static int	NATIONAL_NUMBER	この NumberType は 1 と同じで、発信者が INTERNATIONAL であることを示します。
Static int	NET_SPECIFIC_NUMBER	この NumberType は 2 と同じで、発信者が NATIONAL であることを示します。
Static int	RESERVED_FOR_EXTENSION	この NumberType は 6 と同じで、ファースト ダイヤル コールを示します。現在は使用されていません。
Static int	SUBSCRIBER_NUMBER	この NumberType は 3 と同じで、発信側が MGCP/H.323 ゲートウェイであることを示します。
Static int	UNKNOWN_NUMBER	—

メソッド

表 6-100　CiscoPartyInfo のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getAddress()	アドレスを返します。
boolean	getAddressPI()	アドレスに関連付けられた Presentation Indicator（PI）を返します。true が返された場合、Application はこの Address Name をエンド ユーザに表示できます。false が返された場合、Application はこの Address Name をエンド ユーザに表示しません。
java.lang.String	getDisplayName()	通話者の表示名を返します。
boolean	getDisplayNamePI()	DisplayName に関連付けられた PI を返します。true が返された場合、Application はこの DisplayName をエンド ユーザに表示できます。false が返された場合、Application はこの DisplayName をエンド ユーザに表示しません。
int	getlocale()	通話者の Unicode 表示名のロケールを返します。
int	getNumberType()	通話者の番号種別を返します。
java.lang.String	getUnicodeDisplayName()	通話者の Unicode 表示名を返します。
CiscoUrlInfo	getUrlInfo()	URL 情報を返します。
java.lang.String	getVoiceMailbox()	通話者のボイス メールボックスを返します。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoProvCallParkEv

CiscoProvCallParkEv イベントは、クラスタ内のいずれかのデバイスでコールがパークされたとき、またはパーク解除されたときにプロバイダー オブザーバに配信されます。このイベントを受信するには、CiscoProvider.registerFeature() および CiscoProvFeatureID.MONITOR_CALLPARK_DN を使用して登録を行う必要があります。また、このイベントを受信するには、アプリケーションが使用するユーザ プロファイルで Call Park Retrieval Allowed フラグが有効になっている必要があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, CiscoProvFeatureEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoProvCallParkEv extends CiscoProvFeatureEv

フィールド

表 6-101　CiscoProvCallParkEv のフィールド

インターフェイス	フィールド	説明
Static int	ID	—
Static int	PARK_STATE_ACTIVE	コールがパークされたことを示します。
static int	PARK_STATE_IDLE	コールのパークが解除されたことを示します。
static int	REASON_CALLPARK	コールがパークされると、このイベントが発生することを示します。
static int	REASON_CALLPARKREMAINDER	**推奨されません。** このインターフェイスはスペル ミスのために非推奨になりました。新しいインターフェイス REASON_CALLPARKREMINDER を使用してください。
static int	REASON_CALLPARKREMINDER	コール パーク リマインダの後に、パークしている通話者にコールが戻されることを示します。
static int	REASON_CALLUNPARK	コールのパークが解除されたことを示します。

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-102　CiscoProvCallParkEv のメソッド

インターフェイス	メソッド	説明
int	getintCallIDValue()	このオブジェクトの整数表現を返します。
java.lang.String	getParkDN()	コールがパークされている場所を返します。
java.lang.String	getParkedParty()	パークされた通話者の DN を返します。
java.lang.String	getParkedPartyPartition()	パークされた通話者のパーティションを返します。
java.lang.String	getParkingParty()	パークしている通話者の DN を返します。
java.lang.String	getParkingPartyPartition()	パークしている通話者のパーティションを返します。
java.lang.String	getParkPartyPartition()	パーク DN のパーティションを返します。
int	getReason()	イベントの原因を返します。
int	getState()	コールの状態を返します。Possible states are CiscoProvCallParkEv.PARK_STATE_IDLE CiscoProvCallParkEv.PARK_STATE_ACTIVE.

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoProvFeatureEv から
getFeatureID()

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoProvEv

JTAPI のコアである javax.telephony.events.ProvEv インターフェイスを拡張する CiscoProvEv インターフェイスは、Cisco によって拡張されたすべての JTAPI Provider イベントの基本インターフェイスになります。このパッケージのプロバイダー関連イベントはすべて、直接的または間接的にこのインターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

サブインターフェイス

CiscoAddrActivatedEv, CiscoAddrActivatedOnTerminalEv, CiscoAddrAddedToTerminalEv, CiscoAddrCreatedEv, CiscoAddrRemovedEv, CiscoAddrRemovedFromTerminalEv, CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv, CiscoProvCallParkEv, CiscoProvFeatureEv, CiscoProvTerminalCapabilityChangedEv, CiscoRestrictedEv, CiscoTermActivatedEv, CiscoTermCreatedEv, CiscoTermRemovedEv, CiscoTermRestrictedEv

宣言

public interface CiscoProvEv extends CiscoEv, javax.telephony.events.ProvEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY,

META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

CiscoProvFeatureEv

CiscoProvFeatureEv インターフェイスは、プロバイダー イベントのために com.cisco.jtapi.extensions.CiscoProvEv interface インターフェイスを拡張したものです。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

サブインターフェイス

CiscoProvCallParkEv

宣言

public interface CiscoProvFeatureEv extends CiscoProvEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-103　CiscoProvFeatureEv のメソッド

インターフェイス	メソッド	説明
int	getFeatureID()	アプリケーションがイベントを受け取る機能 ID。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoProvEv を参照してください。
ProvEv.

CiscoProvFeatureID

このインターフェイスは、registerFeature インターフェイスでサポートされている機能をリストします。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoProvFeatureID

フィールド

表 6-104　CiscoProvFeatureID のフィールド

インターフェイス	フィールド	説明
static int	MONITOR_CALLPARK_DN	クラスタ内のいずれかのデバイスでコールがパークまたはパーク解除されたときに、CiscoProvider の registerFeature インターフェイスで CiscoProvCallParkEv を受信するのに使用します。

メソッド

なし

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoProvider

CiscoProvider インターフェイスは、Cisco 固有の機能を追加することによって CiscoProvider インターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoObjectContainer, javax.telephony.Provider

宣言

public interface CiscoProvider extends javax.telephony.Provider, CiscoObjectContainer

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.Provider から
IN_SERVICE, OUT_OF_SERVICE, SHUTDOWN

メソッド

表 6-105　CiscoProvider のメソッド

インターフェイス	メソッド	説明
CiscoTerminal	createTerminal(java.lang.String name)	指定された名前に対応する CiscoTerminal クラスのインスタンスを返します。アプリケーションが十分な機能を備えている必要があります。そうでない場合は、PrivilegeViolationException で CiscoProvider.createTerminal() がスローされます。 **事前条件** this.getState() == Provider.IN_SERVICE **事後条件** Create CiscoTerminal と名前が対応している端末は this.getTerminals() の要素です。 **パラメータ** • name：対象となる CiscoTerminal オブジェクトの名前 **例外** javax.telephony.InvalidArgumentException：指定された名前が、プロバイダーまたはプロバイダーのドメイン内で確認されたどの CiscoMediaTerminal の名前とも対応しません。 javax.telephony.InvalidStateException：プロバイダーがイン サービスではありません。 PreviledgeVoilationException：プロバイダーに十分な機能がありません。CiscoProviderCapabilities.canObserveAnyTerminal() returns false call.getState() == Call.INVALID

表 6-105　CiscoProvider のメソッド（続き）

インターフェイス	メソッド	説明
void	deleteTerminal(CiscoTerminal terminal)	プロバイダー制御から CiscoTerminal オブジェクトを削除します。プロバイダー制御から CiscoTerminal オブジェクトを削除します。アプリケーションが Provider.createTerminal() インターフェイスを使用してこの端末を作成する必要があります。そうでない場合、PreviledgeVoilationException がスローされます。CiscoProvider.deleteTerminal(). **事前条件** this.getState() == Provider.IN_SERVICE **事後条件** 端末のプロバイダー リストから削除された CiscoTerminal オブジェクト。端末は this.getTerminals() の要素ではなくなり、端末に属しているアドレスがが削除されます。 **パラメータ** • terminal：削除する CiscoTerminal オブジェクトへの参照。 **例外** javax.telephony.InvalidArgumentException：提供される端末が this.getTerminals() の要素ではないか、または端末がプロバイダー ドメインではありません。 PrivilegeViolationException：引数で指定された端末が Provider.createTerminal() メソッドを使用して作成された端末ではありません。アプリケーションが削除できるのは、Provider.createTerminal() インターフェイスを使用して作成される端末だけです。
javax.telephony.Address	getAddress(java.lang.String number, java.lang.String partition)	このメソッドで渡される番号とパーティションに対応するアドレス オブジェクトを返します。アドレス オブジェクトは、特定の番号とパーティションの一意の組み合わせです。 **例外** javax.telephony.InvalidArgumentException

表 6-105　CiscoProvider のメソッド （続き）

インターフェイス	メソッド	説明
int	getAppDSCPValue()	CiscoProvider.getAppDSCPValue() を使用して、プロバイダーから DSCP 値を取得します。 **事前条件** this.getState() == Provider.IN_SERVICE **事後条件** このメソッドは、CTI で設定されたアプリケーションの DSCP 値の整数値を返します。
CiscoCall	getCall(CiscoRTPHandle rtpHandle)	特定の端末に関連付けられた RTPHandle のコール オブジェクトを返します。
CiscoCall	getCall(int callleg)	プロバイダー ドメイン内に存在する CiscoCall と、特定の端末に関連付けられた RTPHandle を持つコール オブジェクトを返します。このメソッドは、RTPHandle がどのコールにも関連付けられていない状態になった場合や、このハンドルを含んだ CiscoCallOpenLogicalChannelEv がアプリケーションに送信されたときに端末に callObserver が追加されていなかった場合に、null を返す可能性があります。 **例外** javax.telephony.InvalidStateException
boolean	getCallbackGuardEnabled()	なし
CiscoIntercomAddress[]	getIntercomAddresses()	プロバイダー ドメインに存在している CiscoInterComAddress の配列を返します。

表 6-105　CiscoProvider のメソッド（続き）

インターフェイス	メソッド	説明
CiscoMediaTerminal	getMediaTerminal(java.lang.String name)	指定された名前に対応する CiscoMediaTerminal クラスのインスタンスを返します。各 CiscoMediaTerminal には、JTAPI 実装により固有の名前が割り当てられています。 指定された名前に対する CiscoMediaTerminal がプロバイダーのドメイン内にない場合、このメソッドは、InvalidArgumentException をスローします。 この CiscoMediaTerminal は、Provider.getTerminals() および CiscoProvider.getMediaTerminals() が生成する配列に含まれています。 **事前条件** CiscoMediaTerminal terminal = this.getMediaTerminal(name) を行う。端末は this.getTerminals() の要素です。端末は this.getMediaTerminals() の要素です。 **事後条件** CiscoMediaTerminal terminal = this.getMediaTerminal(name) を行う。端末は this.getTerminals() の要素です。端末は this.getMediaTerminals() の要素です。 **パラメータ** • name：対象となる CiscoMediaTerminal オブジェクトの名前 **例外** javax.telephony.InvalidArgumentException：指定された名前が、プロバイダーまたはプロバイダーのドメイン内で確認されたどの CiscoMediaTerminal の名前とも対応しません。

表 6-105　CiscoProvider のメソッド （続き）

インターフェイス	メソッド	説明
CiscoMediaTerminal[]	getMediaTerminals()	プロバイダーおよびプロバイダーのローカル ドメインに関連付けられた CiscoMediaTerminal の配列を返します。 各 CiscoMediaTerminal には、JTAPI 実装により固有の名前が割り当てられています。 このプロバイダーに関連付けられた CiscoMediaTerminal がない場合、このメソッドは null を返します。 この配列は、Provider.getTerminals() によって返された配列のサブセットです。 **事後条件** CiscoMediaTerminal[] terminals = this.getMediaTerminals() terminals == null または terminals.length >= 1 if terminals != null, terminals is a subset of this.getTerminals () を行う。 **例外** javax.telephony.ResourceUnavailableException：プロバイダーにあるメディア端末の数が、静的配列として返すには大きすぎることを示します。
java.lang.String	getVersion()	なし
void	registerFeature(int featureID)	アプリケーションがプロバイダー イベントを取得する特定の機能を登録します。アプリケーションは、ソフトキーの featureID を渡す必要があります。CiscoProvFeatureID インターフェイスに現在サポートされている機能がリストされています。 **例外** javax.telephony.InvalidStateException javax.telephony.PrivilegeViolationException javax.telephony.InvalidArgumentException

表 6-105　CiscoProvider のメソッド（続き）

インターフェイス	メソッド	説明

Void	setCallbackGuardEnabled(boolean enabled)	オブザーバ コールバックに対して try/catch ロジックを有効または無効にします。オブザーバ コールバックのアプリケーション例外からプロバイダー自体を保護するために、プロバイダーは、通常、アプリケーション インターフェイス（たとえば observer）のすべての呼び出しを、次のコードでガードします。
		`try {`
		`observer.callStateChanged ( ... );`
		`} catch ( Throwable t ) {`
		`// log the exception here`
		`}`
		
		これにより、JTAPI 実装からアプリケーション エラーが隔離されます。JTAPI 実装が処理されない例外を確認し、処理を続行できるため、トラブルシューティングが容易になります。
		
		エラーによっては、回復不能と判断され、JTAPI によって再度スローされます。このため、通常、アプリケーションが終了します。このようなエラーには、ThreadDeath、OutOfMemoryError、および StackOverflowError があります。
		
		JTAPI スレッド内でエラーをトラップする必要があるアプリケーションでは、ThreadGroup のサブクラスを作成し、その ThreadGroup 内のスレッドから JTAPI を初期化する必要があります。
		
		ThreadGroup.uncaughtException () メソッドをオーバーライドすることで、JTAPI スレッドでスローされたすべての回復不能エラーを、アプリケーションで検知可能にできます。場合によっては、JTAPI で積極的にエラーを捕捉することによって、java デバッガでのアプリケーションのトラブルシューティングがより複雑になることがあります。
		
		たとえば、Microsoft Visual J++ バージョン 6.0 では、JTAPI が Throwable を捕捉した場合、アプリケーションのオブザーバ コールバック内のブレークポイントが適正に処理されません。このような場合、JTAPI アプリケーション開発者は、内部の JTAPI try/catch ロジックを無効にすることもできます。

表 6-105　CiscoProvider のメソッド （続き）

インターフェイス	メソッド	説明
		（注） このコールバック ガードを無効にする方法は、アプリケーションのトラブルシューティングを行う場合にだけ使用します。実稼動環境では使用しないでください。デフォルトでは、コールバック ガードは、常に有効にされています。 **パラメータ** • enabled：true の場合、コールバック ガードが有効で、false の場合、コールバック ガードが無効です。
Void	unregisterFeature(int featureID)	特定の機能の登録が解除されます。

継承したメソッド

インターフェイス javax.telephony.Provider から
addObserver, createCall, getAddress, getAddressCapabilities, getAddressCapabilities, getAddresses, getCallCapabilities, getCallCapabilities, getCalls, getCapabilities, getConnectionCapabilities, getConnectionCapabilities, getName, getObservers, getProviderCapabilities, getProviderCapabilities, getState, getTerminal, getTerminalCapabilities, getTerminalCapabilities, getTerminalConnectionCapabilities, getTerminalConnectionCapabilities, getTerminals, removeObserver, shutdown

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から
getObject, setObject

関連資料

なし

CiscoProviderCapabilities

このインターフェイスは、Cisco Unified JTAPI 実装で提供される特定の機能を定義しています。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	メソッド canSupportIPv6()T のサポートを追加します。

スーパーインターフェイス

javax.telephony.capabilities.ProviderCapabilities

宣言

public interface CiscoProviderCapabilities extends javax.telephony.capabilities.ProviderCapabilities

メソッド

表 6-106　CiscoProviderCapabilities のメソッド

インターフェイス	メソッド	説明

boolean	canObserveAnyTerminal()	このメソッドは、ユーザが Cisco Unified Communications Manager でシステムの端末（およびそのアドレス）を監視するための権限をプロビジョニングしたかどうかを確認します。このような端末やアドレスは、JTAPI が起動時に初期化するリストの一部として返されません。このような権限を持つユーザのログインで取得されるプロバイダーは、ProviderCapabilities での canObserverAnyTerminal メソッドから判断できます。ユーザがシステム内の端末を監視できる場合は true を返し、ユーザが制御リスト内の端末およびアドレスだけを監視できる場合は false を返します。
		
		例
		`Provider p = peer.getProvider( loginString );`
		`ProviderCapabilities caps = p.getCapabilities ();`
		`if ( caps instanceof CiscoProviderCapabilities )`
		`{`
		`boolean canObserveAnyTerminal =`
		`((CiscoProviderCapabilities)caps).canObserveAny`
		`Terminal ();`
		`boolean canMonitorParkDN =`
		`((CiscoProviderCapabilities)caps).canMonitorParkDNs`
		`();`
		`boolean canModifyCallingPN=`
		`((CiscoProviderCapabilities)caps).canModifyCalling`
		`Party ();`
		`boolean canRecordCalls =`
		`((CiscoProviderCapabilities)caps).canRecord();`
		`boolean canMonitorCalls =`
		`((CiscoProviderCapabilities)caps).canMonitor();`
		`}`

boolean	canMonitorParkDNs()	このメソッドは、ユーザが Cisco Unified Communications Manager でパーク DN を監視するようにプロビジョニングしたかどうかを確認します。ユーザがパーク DN を監視できる場合は true、そうでない場合は false を返します。

表 6-106　CiscoProviderCapabilities のメソッド （続き）

インターフェイス	メソッド	説明
boolean	canModifyCallingParty()	このメソッドは、ユーザが Cisco Unified Communications Manager でコールの発信側番号を変更するようにプロビジョニングしたかどうかを確認します。ユーザが発信側番号を変更できる場合は true、そうでない場合は false を返します。
boolean	canRecord()	このメソッドは、ユーザが Cisco Unified Communications Manager でコールを録音するようにプロビジョニングしたかどうかを確認します。「Standard CTI Allow Call Recording」ユーザ グループ内のユーザだけがコールを録音できます。ユーザがこのグループに属している場合は True を返します。
boolean	canMonitor()	このメソッドは、ユーザが Cisco Unified Communications Manager でコールをモニタリングするようにプロビジョニングしたかどうかを確認します。「Standard CTI Allow Call Monitoring」ユーザ グループ内のユーザだけがコールのモニタリング要求を開始できます。ユーザがこのグループに属している場合は True を返します。
boolean	canSupportIPv6()	このインターフェイスは、エンタープライズ パラメータ「Enable IPv6」が有効になっている場合は true、そうでない場合は false を返します。

継承したメソッド

インターフェイス javax.telephony.capabilities.ProviderCapabilities から
isObservable

関連資料

canObserveAnyTerminal() を参照してください。

CiscoProviderCapabilityChangedEv

アプリケーション プロバイダー オブザーバは、Cisco Unified Communications Manager でユーザ グループ（capabilitied）にユーザが追加または削除されると、このイベントを受け取ります。このイベントのメソッドでは、変更された機能を確認できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	追加された hasIPv6CapabilityChanged() メソッド。

宣言

public interface CiscoProviderCapabilityChangedEv

フィールド

表 6-107　CiscoProviderCapabilityChangedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし
static int	MODIFY_CGPN	**推奨されません。** この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。
static int	MONITOR_PARKDN	**推奨されません。** この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。
static int	SUPERPROVIDER	**推奨されません。** この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。

メソッド

表 6-108　CiscoProviderCapabilityChangedEv のメソッド

インターフェイス	メソッド	説明
CiscoProviderCapabilities	getCapability()	このメソッドは、ユーザの現在の CiscoProviderCapabilities オブジェクトを返します。

表 6-108　CiscoProviderCapabilityChangedEv のメソッド （続き）

インターフェイス	メソッド	説明
boolean	hasIPv6CapabilityChanged()	このメソッドは、「Enable IPv6」エンタープライズパラメータが変更されたかどうかを判断するために使用できます。 **事前条件** this.getState() == Provider.IN_SERVICE **事後条件** このメソッドは、「Enable IPv6」エンタープライズパラメータが変更された場合に True を返し、そうでない場合は False を返します。
boolean	hasModifyCallingPartyChanged()	このメソッドは、「modify Calling Party」権限が変更されたかどうかを確認します。 **事前条件** provider.getState() == Provider.IN_SERVICE
boolean	hasMonitorCapabilityChanged()	このメソッドは、ユーザのモニタリング機能が変更されたかどうかを確認します。 **事前条件** provider.getState() == Provider.IN_SERVICE
boolean	hasMonitorParkDNChanged()	このメソッドは、「monitor Park DN」権限が変更されたかどうかを確認します。 **事前条件** provider.getState() == Provider.IN_SERVICE
boolean	hasObserveAnyTerminalChanged()	このメソッドは、「can control any terminal」権限が変更されたかどうかを確認します。 **事前条件** provider.getState() == Provider.IN_SERVICE
boolean	hasRecordingCapabilityChanged()	このメソッドは、録音機能が変更されたかどうかを確認します。 **事前条件** provider.getState() == Provider.IN_SERVICE

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoProviderObserver

Provider.addObserver メソッドを使用してプロバイダーを監視する際に、このインターフェイスを実装して CiscoAddrCreatedEv や CiscoTermCreatedEv などの CiscoProvEv イベントを受信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.ProviderObserver

宣言

public interface CiscoProviderObserver extends javax.telephony.ProviderObserver

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.ProviderObserver から
providerChangedEvent

関連資料

CiscoAddrCreatedEv と CiscoTermCreatedEv を参照してください。

CiscoProvTerminalCapabilityChangedEv

このイベントは、端末機能が変更された場合にプロバイダーに送信されます。このイベントは、アプリケーション オブザーバで提供されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.0(1)	イベントが追加されました。
7.0(1)	CiscoMediaTerminals または CiscoRouteTerminals だけが返されるように CiscoTerminal[] が変更されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoProvTerminalCapabilityChangedEv extends CiscoProvEv

フィールド

表 6-109　CiscoProvTerminalCapabilityChangedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-110　CiscoProvTerminalCapabilityChangedEv のメソッド

インターフェイス	メソッド	説明
CiscoTerminal[]	getTerminals()	機能が変更された CiscoTerminals の配列を返します。Cisco Unified Communications Manager Release 7.0(1) では、CiscoMediaTerminals または CiscoRouteTerminals だけが返されるように、CiscoTerminal[] インターフェイスがモニタリングされます。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

なし

CiscoRecorderInfo

このインターフェイスは、録音セッションにおける録音側に関する情報を提供します。録音セッションがアクティブな場合、このインターフェイスは、録音デバイスに関する情報を提供します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRecorderInfo

フィールド

なし

メソッド

表 6-111　CiscoRecorderInfo のメソッド

インターフェイス	メソッド	説明
CiscoAddress	getAddress()	録音側のアドレスを返します。
java.lang.String	getTerminalName()	録音デバイスの端末名を返します。

関連資料

なし

CiscoRestrictedEv

CiscoRestrictedEv イベントは、CiscoAddrRestrictedEv および CiscoAddrRestrictedOnTerminalEv イベントの親クラスです。これは制限されたイベントの基底クラスであり、すべての制限されたイベントの原因コードを定義します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

サブインターフェイス

CiscoAddrRestrictedEv, CiscoAddrRestrictedOnTerminalEv

宣言

public interface CiscoRestrictedEv extends CiscoProvEv

フィールド

表 6-112　CiscoRestrictedEv のフィールド

インターフェイス	フィールド	説明
static int	CAUSE_UNKNOWN	不明な理由のために端末が制限されています。
static int	CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION	端末はサポートされない設定（たとえば、ロールオーバー オプション）のために制限されています。
static int	CAUSE_UNSUPPORTED_PROTOCOL	制御リスト内の端末は Cisco Unified JTAPI でサポートされていないプロトコルを使用しています。
static int	CAUSE_USER_RESTRICTED	端末またはアドレスが制限とマークされています。
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

なし

CiscoRouteAddress

このインターフェイスは推奨されません。また、実装されていません。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.Address, javax.telephony.callcenter.RouteAddress

宣言

public interface CiscoRouteAddress extends javax.telephony.callcenter.RouteAddress

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.callcenter.RouteAddress から
ALL_ROUTE_ADDRESS

メソッド

表 6-113　CiscoRouteAddress のメソッド

インターフェイス	メソッド	説明
void	registerRouteCallback(javax.telephony.callcenter.RouteCallback routeCallback, boolean disableAutoRehoming)	推奨されません。 **例外** javax.telephony.ResourceUnavailableException javax,telephony.MethodNotSupportedException

継承したメソッド

インターフェイス javax.telephony.callcenter.RouteAddress から
cancelRouteCallback, getActiveRouteSessions, getRouteCallback, registerRouteCallback

インターフェイス javax.telephony.Address から
addCallObserver, addObserver, getAddressCapabilities, getCallObservers, getCapabilities, getConnections, getName, getObservers, getProvider, getTerminals, removeCallObserver, removeObserver

関連資料

なし

CiscoRouteEvent

CiscoRouteEvent インターフェイスは、Cisco 固有の機能を追加することによって RouteEvent インターフェイスを拡張します。getCallingPartyIpAddr メソッドを使用して、発信側デバイスの IP アドレスを取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	getCallingPartyIpAddr_v6() メソッドが追加されました。

スーパーインターフェイス

javax.telephony.callcenter.events.RouteEvent, javax.telephony.callcenter.events.RouteSessionEvent

宣言

public interface CiscoRouteEvent extends javax.telephony.callcenter.events.RouteEvent

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.callcenter.events.RouteEvent から
SELECT_ACD, SELECT_EMERGENCY, SELECT_LEAST_COST, SELECT_NORMAL, SELECT_USER_DEFINED

メソッド

表 6-114　CiscoRouteEvent のメソッド

インターフェイス	メソッド	説明
java.net.InetAddress	getCallingPartyIpAddr_v6()	発信側の IPv6 アドレスを返します。IP アドレスが取得できない場合、このメソッドは IP アドレス 0::0 の InetAddress と、null のホスト名を返します。このオブジェクトを出力すると、「null/0::0」の文字列表現が出力されます。戻り値：InetAddress。
java.net.InetAddress	getCallingPartyIpAddr()	発信側の IP アドレスを返します。IP アドレスが取得できない場合、このメソッドは IP アドレス 0.0.0.0 の InetAddress と、null の ホスト名を返します。このオブジェクトを出力すると、「null/0.0.0.0」の文字列表現が出力されます。

継承したメソッド

インターフェイス javax.telephony.callcenter.events.RouteEvent から
getCallingAddress, getCallingTerminal, getCurrentRouteAddress, getRouteSelectAlgorithm, getSetupInformation

インターフェイス javax.telephony.callcenter.events.RouteSessionEvent から
getRouteSession

関連資料

なし

CiscoRouteSession

CiscoRouteSession インターフェイスは、RouteSession に関連付けられた、基になるコールにアプリケーションがアクセスすることをサポートします。また、このインターフェイスは RouteEndEvent のさまざまな内部エラーを表示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcenter.RouteSession

宣言

public interface CiscoRouteSession extends javax.telephony.callcenter.RouteSession

フィールド

表 6-115　CiscoRouteSession のフィールド

インターフェイス	フィールド	説明
static final int	ERROR_ROUTESELECT_TIMEOUT	各 routeEvent() または reRouteEvent() が送られるたびに、routeSelect() または endRoute() で応答するアプリケーションのタイマーが始動します。このタイマーのデフォルト値は、5 秒です。この時間内にアプリケーションが応答しない場合、このエラーで endRoute が呼び出されます。
static final int	ERROR_NO_CALLBACK	デフォルト ルート メカニズムがないので、このアプリケーションにコールバックが登録されていない場合は、このエラーが設定された endRoute が呼び出されます。
static final int	ERROR_INVALID_STATE	ルーティング中に、内部的な InvalidStateException が発生した場合、または事前条件または事後条件の一部が満たされなかった場合には、このエラーで endRoute が呼び出されます。
static final int	DEFAULT_SEARCH_SPACE	この実装のデフォルトの検索スペースを使用してリダイレクトを実行します。デフォルトでは、発側の検索スペースを使用します。

表 6-115　CiscoRouteSession のフィールド （続き）

インターフェイス	フィールド	説明
static final int	CALLINGADDRESS_SEARCH_SPACE	発信元アドレスの検索スペースを使用してリダイレクトを実行する必要があることを示します。
static final int	ROUTEADDRESS_SEARCH_SPACE	ルート ポイント アドレスの検索スペースを使用して、リダイレクトを実行する必要があることを示します。
static final int	DONOT_RESET_ORIGINALCALLED	これは PreferedOriginalCalled Option のパラメータ値で、OriginalCalled を再設定しないことを指定します。
static final int	RESET_ORIGINALCALLED	これは PreferedOriginalCalled のパラメータ値で、preferredOriginalCalledOption の値がこの値に設定された場合、OriginalCalled は preferredOriginalCalledNumber にリセットされます。
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_NEEDED	RouteSession.getCause() で返されるこの定数は、selectRoute の routeSelectedElement に必要な FAC コードが含まれていないことを示します。
static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_NEEDED	RouteSession.getCause() で返されるこの定数は、selectRoute の routeSelectedElement に必要な CMC コードが含まれていないことを示します。
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	RouteSession.getCause() で返されるこの定数は、selectRoute の routeSelectedElement に必要な FAC コードおよび CMC コードが含まれていないことを示します。
static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_INVALID	RouteSession.getCause() で返されるこの定数は、selectRoute の routeSelectedElement に無効な FAC コードが含まれていることを示します。
static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_INVALID	RouteSession.getCause() で返されるこの定数は、selectRoute の routeSelectedElement に有効な CMC コードが含まれていることを示します。

継承したフィールド

インターフェイス javax.telephony.callcenter.RouteSession から
CAUSE_INVALID_DESTINATION, CAUSE_NO_ERROR, CAUSE_PARAMETER_NOT_SUPPORTED, CAUSE_ROUTING_TIMER_EXPIRED, CAUSE_STATE_INCOMPATIBLE, CAUSE_UNSPECIFIED_ERROR, ERROR_RESOURCE_BUSY, ERROR_RESOURCE_OUT_OF_SERVICE, ERROR_UNKNOWN, RE_ROUTE, ROUTE, ROUTE_CALLBACK_ENDED, ROUTE_END, ROUTE_USED

メソッド

表 6-116　CiscoRouteSession のメソッド

インターフェイス	メソッド	説明
javax.telephony.Call	getCall()	この RouteSession. に関連付けられたコールを返します。
void	selectRoute (java.lang.String[] routeSelected, int callingSearchSpace)	RouteSession インターフェイスの selectRoute メソッドをオーバーロードして、コールがルートの宛先にリダイレクトされるときに使用する、発信元検索スペースをアプリケーションで指定できるようにします。 **例外** javax.telephony.MethodNotSupportedException **パラメータ** • callingSearchSpace：CiscoRouteSession.DEFAULT_SEARCH_SPACE、CiscoRouteSession.CALLINGADDRESS_SEARCH_SPACE、または CiscoRouteSession.ROUTEADDRESS_SEARCH_SPACE のいずれか。 • routeSelected：コールに対して可能な宛先のリスト。

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
void	selectRoute (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] modifyingCallingNumber)	発番号が変更されたコールをルーティング可能な 1 つ以上の宛先を選択します。このメソッドは、宛先の電話のアドレス名と、modifyingCallingNumber の文字列配列を優先順位に従って引数をとります。 最も優先順位の高い宛先が指定配列の最初の要素になり、ルーティングは、この宛先から順に、対応する発信者番号変更の要素を使用して試みられます。 modifiedCallingNumber 要素が null である場合、コールがその特定の routeSelected 要素にルーティングされた際に発信者番号が変更されません。宛先が正常に選択されるまで、指定された宛先アドレスを順に使用するように試行します。コールが宛先にルーティングされると、RouteUsedEvent がアプリケーションに配信されます。 **事前条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == RouteSession.ROUTE or RouteSession.RE_ROUTE **事後条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == • RouteSession.ROUTE_USED（コールが正常にルーティングされた場合）。正常な宛先が選択されると、この RouteSession に RouteUsedEvent が配信されます。 **パラメータ** • routeSelected：callcallingSearchSpace に対して可能な宛先は、CiscoRouteSession.DEFAULT_SEARCH_SPACE、CiscoRouteSession.CALLINGADDRESS_SEARCH_SPACE、または CiscoRouteSession.ROUTEADDRESS_SEARCH_SPACE です。 • modifyingCallingNumber：コールが routeSelected 要素に達したときにアプリケーションが発番号を変更する要素の配列です。 **例外** • com.cisco.jtapi.MethodNotSupportedExceptionImpl（この実装では、ルーティングをサポートしません） • javax.telephony.PrivilegeViolationException（ユーザに、このメソッドを使用するために必要な権限がありません） • javax.telephony.MethodNotSupportedException selectRoute

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
void	selectRoute (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption	コールをルーティング可能な 1 つ以上の宛先を選択します。このメソッドは、宛先の電話のアドレス名の文字列配列（優先順位順）と、PreferredOriginalCalled 番号の文字列配列を引数にとります。 PreferedOriginalCalled 番号は、宛先電話名配列のインデックスに基づいて選択されます。宛先配列に対応するインデックスが、PreferedOriginalCalled 番号配列にない場合は、preferedOriginalCalled が宛先に設定されます。 最も優先順位の高い宛先が指定配列の最初の要素になり、ルーティングは、この宛先から連続して試みられます。コールが正常にルーティングされるまで、指定された各宛先アドレスが連続して試行されます。選択されたルーティングの宛先が成功し、コールがその宛先にルーティングされると、RouteUsedEvent イベントがアプリケーションに配信されます。 **事前条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == RouteSession.ROUTE or • RouteSession.RE_ROUTE **事後条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == • RouteSession.ROUTE_USED（コールが正常にルーティングされた場合）。正常な宛先が選択されると、この RouteSession に RouteUsedEvent が配信されます。 **パラメータ** • routeSelected：コールに対して可能な宛先。 • preferedOriginalCalledNumber：routeSelected リストの一致する配列インデックスのルートに対応する各項目のリスト。 • preferedOriginalCalledOption：routeSelected リストに対応する各オプションのリスト。このオプションは、OriginalCalled を preferedOriginalCalledNumber に設定するかどうかを指定します。オプションの値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED および CiscoRouteSession.RESET_ORIGINALCALLED です。値が指定されていないか null の場合、デフォルト値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED です。

表 6-116　CiscoRouteSession のメソッド（続き）

インターフェイス	メソッド	説明
		例外 com.cisco.jtapi.MethodNotSupportedExceptionImpl（この実装では、ルーティングをサポートしません） javax.telephony.PrivilegeViolationException（ユーザに、このメソッドを使用するために必要な権限がありません） javax.telephony.MethodNotSupportedException selectRoute
void	selectRoute (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] modifyingCallingNumber, java.lang.String[] prefered OriginalCalledNumber, int[] preferedOriginal CalledOption, java.lang.String[] facCode, java.lang.String[] cmcCode)	ルーティング可能な 1 つ以上の宛先を選択します。次の文字列配列を引数にとります。 • 宛先の電話のアドレス名（優先順位順） • PreferredOriginalCalled 番号 • FAC • CMC PreferedOriginalCalled 番号は、宛先電話名配列のインデックスに対応して選択されます。インデックスが referedOriginalCalled 番号配列にない場合は、preferedOriginalCalled が宛先に設定されます。 最も優先順位の高い宛先が指定配列の最初の要素になり、ルーティングは、この宛先から順に試みられます。コールが正常にルーティングされるまで、指定された宛先アドレスを順に使用するように試行します。選択されたルーティングの宛先が成功し、コールがその宛先にルーティングされると、RouteUsedEvent イベントがアプリケーションに配信されます。 **事前条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == RouteSession.ROUTE • RouteSession.RE_ROUTE **事後条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == • RouteSession.ROUTE_USED（コールが正常にルーティングされた場合）。正常な宛先が選択されると、この RouteSession に RouteUsedEvent が配信されます。 **パラメータ** • routeSelected：可能な宛先のリスト。 • preferedOriginalCalledNumber：routeSelected リスト内の同じ配列インデックスのルートに対応する各番号のリスト。

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
		• list.preferedOriginalCalledOption：RouteList に対応する各オプションのリスト。このオプションは、OriginalCalled を preferedOriginalCalledNumber に設定するかどうかを指定します。オプションの値は iscoRouteSession.DONOT_RESET_ORIGINALCALLED および CiscoRouteSession.RESET_ORIGINALCALLED です。値が指定されていないか null の場合、デフォルト値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED です。 • modifyingCallingNumber：コールがルート選択要素に達したときにアプリケーションが発番号を変更する要素の配列です。アプリケーションが発番号を変更しない場合は、アプリケーションによってこのパラメータに null 値が渡される必要があります。 • facCode (Forced Authorization Code [FAC])：routeSelected 要素に FAC が必要な場合、対応する facCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 • cmcCode：（Client Matter Code [CMC]）routeSelected 要素に CMC が必要な場合、対応する cmcCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 **例外** • com.cisco.jtapi.MethodNotSupportedExceptionImpl（この実装では、ルーティングをサポートしません） • javax.telephony.PrivilegeViolationException（ユーザに、このメソッドを使用するために必要な権限がありません） • javax.telephony.MethodNotSupportedException selectRoute

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
void	selectRoute (java.lang.String[] routeSelected, int callingSearchSpace, java.lang.String[] modifyingCallingNumber, java.lang.String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption, java.lang.String[] facCode, java.lang.String[] cmcCode, int featurePriority)	ルーティング可能な 1 つ以上の宛先を選択します。次の文字列配列を引数にとります。 • 宛先の電話のアドレス名（優先順位順） • PreferredOriginalCalled 番号 • FAC • CMC • 整数の優先順位 PreferedOriginalCalled 番号は、宛先電話名配列のインデックスに基づいて選択されます。インデックスが referedOriginalCalled 番号配列にない場合は、preferedOriginalCalled が宛先に設定されます。 最も優先順位の高い宛先が指定配列の最初の要素になり、ルーティングは、この宛先から連続して試みられます。コールが正常にルーティングされるまで、指定された宛先アドレスを使用するように連続して試行します。選択されたルーティングの宛先が成功し、コールがその宛先にルーティングされると、RouteUsedEvent イベントがアプリケーションに配信されます。 **事前条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == RouteSession.ROUTE • RouteSession.RE_ROUTE **事後条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == • RouteSession.ROUTE_USED（コールが正常にルーティングされた場合）。 **パラメータ** routeSelected：コールに対して可能な宛先。 preferedOriginalCalledNumber：routeSelected リスト内の同じ配列インデックスのルートに対応する各要素のリスト。 preferedOriginalCalledOption：RouteList に対応する各オプションのリスト。このオプションは、OriginalCalled を preferedOriginalCalledNumber に設定するかどうかを指定します。オプションの値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED および CiscoRouteSession.RESET_ORIGINALCALLED です。値が指定されていないか null の場合、デフォルト値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED です。

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
		• modifyingCallingNumber：コールが routeselected 要素に達したときにアプリケーションが発番号を変更する要素の配列です。アプリケーションが発番号を変更しない場合は、アプリケーションによってこのパラメータに null 値が渡される必要があります。 • facCode (Forced Authorization Code [FAC]) : routeSelected 要素に FAC が必要な場合、対応する facCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 • cmcCode (Client Matter Code [CMC]) : routeSelected 要素に CMC が必要な場合、対応する cmcCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 • featurePriority：コールの機能の優先順位を設定します。特定の優先順位を設定しない場合、CiscoCall.FEATUREPRIORITY_NORMAL が設定されます。featurePriority パラメータは、次のいずれかになります。 – CiscoCall.FEATUREPRIORITY_NORMAL – CiscoCall.FEATUREPRIORITY_URGENT – CiscoCall.FEATUREPRIORITY_EMERGENCY **例外** • javax.telephony.PrivilegeViolationException（ユーザに、このメソッドを使用するために必要な権限がありません） • com.cisco.jtapi.MethodNotSupportedExceptionImpl（この実装では、ルーティングをサポートしません） • javax.telephony.MethodNotSupportedException

表 6-116　CiscoRouteSession のメソッド（続き）

インターフェイス	メソッド	説明
void	selectRoute(java.lang.String[] routeSelected, int[] callingSearchSpace, java.lang.String[] modifyingCallingNumber, java.lang.String[] preferedOriginalCalledNumber, int[] preferedOriginalCalledOption, java.lang.String[] facCode, java.lang.String[] cmcCode, int[] featurePriority)	ルーティング可能な 1 つ以上の宛先を選択します。次の文字列配列を引数にとります。 • 宛先の電話のアドレス名（優先順位順） • コーリング サーチ スペース • modifyingCallingNumbers • PreferredOriginalCalled 番号 • FAC • CMC • 機能プライオリティ PreferedOriginalCalled 番号は、宛先電話名配列のインデックスに基づいて選択されます。インデックスが referedOriginalCalled 番号配列にない場合は、preferedOriginalCalled が宛先に設定されます。 最も優先順位の高い宛先が指定配列の最初の要素になり、ルーティングは、この宛先から連続して試みられます。コールが正常にルーティングされるまで、指定された宛先アドレスを使用するように連続して試行します。 選択されたルーティングの宛先が成功し、コールがその宛先にルーティングされると、RouteUsedEvent イベントがアプリケーションに配信されます。 **事前条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == RouteSession.ROUTE • RouteSession.RE_ROUTE **事後条件** • this.getRouteAddress().getProvider().getState() == • Provider.IN_SERVICE this.getState() == • RouteSession.ROUTE_USED（コールが正常にルーティングされた場合）。 **パラメータ** • routeSelected：コールに対して可能な宛先のリスト。 • callingSearchSpace：選択したルートごとに、CiscoRouteSession.DEFAULT_SEARCH_SPACE、CiscoRouteSession.CALLINGADDRESS_SEARCH_SPACE、または CiscoRouteSession.ROUTEADDRESS_SEARCH_SPACE です。

表 6-116　CiscoRouteSession のメソッド （続き）

インターフェイス	メソッド	説明
		• preferedOriginalCalledNumber : routeSelected リスト内の同じ配列インデックスのルートに対応する各要素のリスト。 • preferedOriginalCalledOption : RouteList に対応する各オプションのリスト。このオプションは、OriginalCalled を preferedOriginalCalledNumber に設定するかどうかを指定します。オプションの値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED および CiscoRouteSession.RESET_ORIGINALCALLED です。値が指定されていないか null の場合、デフォルト値は CiscoRouteSession.DONOT_RESET_ORIGINALCALLED です。 • modifyingCallingNumber : コールが routeselected 要素に達したときにアプリケーションが発番号を変更する要素の配列です。アプリケーションが発番号を変更しない場合は、アプリケーションによってこのパラメータに null 値が渡される必要があります。 • facCode (Forced Authorization Code [FAC]) : routeSelected 要素に FAC が必要な場合、対応する facCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 • cmcCode (Client Matter Code [CMC]) : routeSelected 要素に CMC が必要な場合、対応する cmcCode 要素にそのコードを含める必要があります。routeSelected 要素にコードが不要な場合、アプリケーションはこのパラメータに null 値を渡す必要があります。 • featurePriority : 選択したルートごとに、機能プライオリティを設定できます。特定の優先順位を設定しない場合、CiscoCall.FEATUREPRIORITY_NORMAL が設定されます。featurePriority パラメータは、CiscoCall.FEATUREPRIORITY_NORMAL、CiscoCall.FEATUREPRIORITY_URGENT、または CiscoCall.FEATUREPRIORITY_EMERGENCY のいずれかになります。 **例外** • javax.telephony.PrivilegeViolationException（ユーザに、このメソッドを使用するために必要な権限がありません） • com.cisco.jtapi.MethodNotSupportedExceptionImpl（この実装では、ルーティングをサポートしません） • javax.telephony.MethodNotSupportedException

継承したメソッド

インターフェイス javax.telephony.callcenter.RouteSession から
endRoute, getCause, getRouteAddress, getState, selectRoute

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoRouteTerminal

CiscoRouteTerminal は、アプリケーションによる RTP メディア ストリームの終端を可能にする、特殊な CiscoTerminal です。CiscoRouteTerminal は、通常の CiscoTerminal と異なり、物理的なテレフォニー エンドポイントではないため、サードパーティ方式で監視、制御することができます。CiscoRouteTerminal は論理的なテレフォニー エンドポイントなので、コールをルーティングしメディアを終端する必要があるアプリケーションに関連付けることができます。CiscoMediaTerminal とは異なり、CiscoRouteTerminal では複数のアクティブ コールを同時に保持できます。通常、アプリケーションは、エージェントが次の発信者に応対できるようになるまでの間、コールをキューに格納するために使用されます。

（注） CiscoRouteTerminal は Cisco Unified Communications Manager の CTI Route Point です。

メディアの終端プロセスには次の 3 つの段階があります。

ステップ 1　アプリケーションが、CiscoRouteTerminal.register メソッドを使用して、メディア機能をこの端末に登録します。

ステップ 2　CiscoTerminalObserver インターフェイスを実装するオブザーバを、アプリケーションが Terminal.addObserver メソッドを使用して追加します。

ステップ 3　アプリケーションは CiscoRouteTerminal または CiscoRouteAddress で addCallObserver を呼び出して、コールを着信して応答する必要があります。

アプリケーションはコールごとに CiscoMediaOpenLogicalChannelEv を受信します。メディアが停止した場合は、再確立する必要があります。アプリケーションは、CiscoRouteTerminal の setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。

（注） **重要**：CiscoJtapiClient 1.4 リリース以前で作成されたすべてのアプリケーションは、メディア終端を必要としない場合、修正して CiscoRouteTerminal.NO_MEDIA_TERMINATION タイプに登録する必要があります。

登録されるメディア機能および登録タイプが同じであれば、複数のアプリケーションを同じ RoutePoint に登録できます。CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION に登録された場合は、callObserver が追加されたときに CiscoMediaOpenLogicalChannelEv を受信します。ただし、1 つのアプリケーションだけが setRTPParams を呼び出すことができます。

メディア終端を行うアプリケーションは、RouteAddress または CiscoRouteTerminals に CallObserver を追加する必要があります。アプリケーションは動的タイプで登録せず、registerRouteCallBack を追加する必要があります。メディア終端を行わない場合、アプリケーションは、registerRouteCallBack だけを使用する必要があります。アプリケーションは registerRouteCallBack と callObserver を同時に追加できません。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	activeAddressingMode に次のモードが追加されました。 • CiscoTerminal.IP_ADDRESSING_MODE_IPv6 • CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6

スーパーインターフェイス

CiscoObjectContainer, CiscoTerminal, javax.telephony.Terminal

宣言

public interface CiscoRouteTerminal extends CiscoTerminal

フィールド

表 6-117　CiscoRouteTerminal のフィールド

インターフェイス	フィールド	説明
static int	DYNAMIC_MEDIA_REGISTRATION	メディア終端を行うアプリケーションは、登録要求でこのタイプを登録し、サポートしている機能を渡す必要があります。
static int	NO_MEDIA_REGISTRATION	メディア終端を行わないアプリケーションは、登録要求でこのタイプを登録し、CiscoMediaCapability に null 値を渡す必要があります。 registrationType が CiscoRouteTerminal.NO_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端できず、CiscoRouteTerminal をコールのルーティング用に使用できます。

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoTerminal から
ASCII_ENCODING, DEVICESTATE_ACTIVE, DEVICESTATE_ALERTING, DEVICESTATE_HELD, DEVICESTATE_IDLE, DEVICESTATE_UNKNOWN, DEVICESTATE_WHISPER, DND_OPTION_CALL_REJECT, DND_OPTION_NONE,

DND_OPTION_RINGER_OFF, IN_SERVICE, IP_ADDRESSING_MODE_IPV4, IP_ADDRESSING_MODE_IPV4_V6, IP_ADDRESSING_MODE_IPV6, IP_ADDRESSING_MODE_UNKNOWN, IP_ADDRESSING_MODE_UNKNOWN_ANATRED, NOT_APPLICABLE, OUT_OF_SERVICE, UCS2UNICODE_ENCODING, UNKNOWN_ENCODING

メソッド

表 6-118　CiscoRouteTerminal のメソッド

インターフェイス	メソッド	説明
void	register (CiscoMediaCapability[] capabilities, int registrationType)	指定した CiscoMediaCapabilities と登録タイプで端末を登録します。プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。CiscoRouteTerminal が登録されると、正常に終了します。 **パラメータ** • capabilities：アプリケーションがこの端末に対してサポートする RTP 符号化方式のリスト。アプリケーションでメディア終端に関する処理をしない場合は null 値を渡すことができます。 • registrationType：CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION または CiscoRouteTerminal.NO_MEDIA_REGISTRATION registrationType が CiscoRouteTerminal.NO_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端できず、CiscoRouteTerminal をコールのルーティング用に使用できます。 registrationType が CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端でき、アクティブ コールを複数持つことができます。この registrationType は、アプリケーションが各コールに対して動的に IP アドレスおよびポートを指定することを示します。この種類で登録するアプリケーションは、コールごとに CiscoMediaOpenLogicalChannelEv を受信するので、CiscoRouteTerminal の setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。 **例外** • CiscoRegistrationException

表 6-118　CiscoRouteTerminal のメソッド （続き）

インターフェイス	メソッド	説明
void	register (CiscoMediaCapability[] capabilities, int registrationType, int[] algorithmIDs)	指定された CiscoMediaCapabilities、registrationType およびサポートされる SRTP アルゴリズムで端末を登録します。プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。CiscoRouteTerminal が登録されると、このメソッドは正常に終了します。 パラメータ • capabilities：アプリケーションがこの端末に対してサポートする RTP 符号化方式のリスト。アプリケーションでメディア終端に関する処理をしない場合は null 値を渡すことができます。 • registrationType：CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION または CiscoRouteTerminal.NO_MEDIA_REGISTRATION registrationType が CiscoRouteTerminal.NO_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端できず、CiscoRouteTerminal をコールのルーティング用に使用できます。このメソッドのその他のパラメータは無視されます。 registrationType が CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端でき、アクティブ コールを複数持つことができます。この registrationType は、アプリケーションが各コールに対して動的に IP アドレスおよびポートを指定することを示します。この種類で登録するアプリケーションは、コールごとに CiscoMediaOpenLogicalChannelEv を受信するので、setRTPParams メソッドを使用して、IP アドレスおよびポート番号を提示する必要があります。 • algorithmIDs：アプリケーションがこの端末に対してサポートする SRTP アルゴリズムのリスト。これを使用するには、アプリケーションで TLS Link および SRTP Enabled フラグを有効にする必要があります。AlgorithmID は、CiscoMediaEncryptionSupportedAlgorithms のいずれかにする必要があります。 **例外** • javax.telephony.PrivilegeViolationException （アプリケーションはこのメソッドを使用しようとしますが、使用するための権限がありません）。 • CiscoRegistrationException

表 6-118　CiscoRouteTerminal のメソッド （続き）

インターフェイス	メソッド	説明
void	register (CiscoMediaCapability[] capabilities, int registrationType, int[] algorithmIDs, int activeAddressingMode)	CiscoRouteTerminal は CiscoTerminal.UNREGISTERED 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このメソッドが成功した場合は、RouteTerminal が登録されます。指定された CiscoMediaCapabilities 、登録タイプおよびサポートされる SRTP アルゴリズムで端末を登録します。 registrationType が CiscoRouteTerminal.NO_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端できず、ルート ポイントをコールのルーティング用に使用できます。このメソッドのその他のパラメータは無視されます。 registrationType が CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION である場合、アプリケーションはメディアを終端でき、アクティブ コールを複数持つことができます。これは、アプリケーションが各コールに対して動的に ipAddress およびポートを指定することを示します。 このタイプに登録するアプリケーションは、各コールに対して CiscoMediaOpenLogicalChannelEv を受信し、CiscoRouteTerminal に対する setRTPParams メソッドを使用して ipAddress とポート番号を指定する必要があります。 **メソッドの引数** アプリケーションがこの端末に対してサポートする RTP 符号化方式の種類を示します。アプリケーションでメディア終端に関する処理をしない場合は null 値を渡すことができます。registrationType は CiscoRouteTerminal.NO_MEDIA_REGISTRATION または CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION です。 サポートされるアルゴリズムはアプリケーションがこの端末に対してサポートする SRTP アルゴリズムです。これを使用するには、アプリケーションで TLS Link および SRTP Enabled フラグを有効にする必要があります。アプリケーションにこのメソッドを使用する権限がない場合、PrivilegeViolationException がスローされます。 **事後条件** CiscoRouteTerminal が登録されると、このメソッドは正常に終了します。 **パラメータ** • capabilities：この端末でサポートされている RTP 符号化方式のリスト。 • registrationType：CiscoRouteTerminal.DYNAMIC_MEDIA_REGISTRATION または CiscoRouteTerminal.NO_MEDIA_REGISTRATION • algorithmIDs：サポートされる SRTP アルゴリズムのリスト。AlgorithmID は、CiscoMediaEncryption SupportedAlgorithms のいずれか 1 つだけになります。

表 6-118　CiscoRouteTerminal のメソッド （続き）

インターフェイス	メソッド	説明
		• activeAddressingMode：アプリケーションがこの CiscoRouteTerminal を登録する IP アドレッシング モード モードは次のいずれかです。 – CiscoTerminal.IP_ADDRESSING_MODE_IPv4 – CiscoTerminal.IP_ADDRESSING_MODE_IPv6 – CiscoTerminal.IP_ADDRESSING_MODE_IPv4_v6 **例外** • CiscoRegistrationException • javax.telephony.PrivilegeViolationException
void	unregister()	CiscoRouteTerminal は登録されている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このメソッドが成功した場合は、CiscoRouteTerminal の登録が解除されます。 **事後条件** • MediaTerminal の登録が解除されると、このメソッドは正常に終了します。 **例外** • CiscoUnregistrationException
void	setRTPParams(CiscoRTPHandle rtpHandle, CiscoRTPParams rtpParams)	アプリケーションは、IP アドレスおよび RTP ポート番号を設定して、コールのメディア ストリームを動的に行います。これを行うには、機能だけを指定して MediaTerminal または CiscoRouteTeminal を登録する必要があります。 アプリケーションでは、TerminalObserver で CiscoCallOpenLogicalChannelEv を受信したときに、このメソッドを起動する必要があります。 **パラメータ** rtpHandle：アプリケーションが CiscoCallOpenLogicalChannelEvrtpParams で受け取るハンドル。CiscoRTPParams を参照してください。 **例外** • javax.telephony.InvalidStateException • javax.telephony.InvalidArgumentException • javax.telephony.PrivilegeViolationException
boolean	isRegistered()	CiscoMediaTerminal が登録されている場合は true、そうでない場合は false を返します。CiscoRouteTerminal が OutOfService である場合は false、InService である場合は true を返します。CTIManager の障害の場合は、false を返します。

表 6-118　CiscoRouteTerminal のメソッド （続き）

インターフェイス	メソッド	説明
boolean	isRegisteredByThisApp()	このメソッドは、このアプリケーションが登録要求を発行して成功した場合に true を返します。この登録は、CTIManager の障害によりデバイスがアウトオブサービス状態にある場合でも有効です。これは、このアプリケーションがデバイスの登録を解除するまで true を返します。
int	getIPAddressingMode()	アプリケーションはこの API を起動して、CiscoRouteTerminal の IP アドレッシング モードを照会できます。アドレッシング モードは次のいずれかの定数になります。 • CiscoTerminal.IP_ADDRESSING_IPv4 • CiscoTerminal.IP_ADDRESSING_IPv6 • CiscoTerminal.IP_ADDRESSING_IPv4_v6

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoTerminal から

createSnapshot, getAltScript, getDeviceState, getDNDOption, getDNDStatus, getEMLoginUsername, getFilter, getLocale, getProtocol, getRegistrationState, getRTPInputProperties, getRTPOutputProperties, getState, getSupportedEncoding, isRestricted, sendData, sendData, setDNDStatus, setFilter, unPark

インターフェイス javax.telephony.Terminal から

addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から

getObject, setObject

関連資料

CiscoTerminal と 「定数フィールド値」（P.F-1）を参照してください。
CiscoMediaOpenLogicalChannelEv

CiscoRouteUsedEvent

CiscoRouteUsedEvent イベントは、RouteSession が RouteSession.ROUTE_USED 状態に移行し、アプリケーションによるルーティングの結果、コールが宛先で終端したことを示します。このインターフェイスは RouteUsedEvent インターフェイスを拡張し、RouteCallback インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcenter.events.RouteSessionEvent,
javax.telephony.callcenter.events.RouteUsedEvent

宣言

public interface CiscoRouteUsedEvent extends javax.telephony.callcenter.events.RouteUsedEvent

フィールド

なし

メソッド

表 6-119　CiscoRouteUsedEvent のメソッド

インターフェイス	メソッド	説明
Int	getRouteSelectedIndex()	コールのルーティング先となるルートの配列インデックスを返します。

継承したメソッド

インターフェイス javax.telephony.callcenter.events.RouteUsedEvent から
getCallingAddress, getCallingTerminal, getDomain, getRouteUsed

インターフェイス javax.telephony.callcenter.events.RouteSessionEvent から
getRouteSession

関連資料

RouteSession、RouteCallback および RouteSessionEvent を参照してください。

CiscoRTPBitRate

RTPBitRate インターフェイスには、G.723 RTP ビット レートを記述する定数が入ります。CiscoRTPInputProperties.getBitRate と CiscoRTPOutputProperties.getBitRate はこれらの定数を返します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPBitRate

フィールド

表 6-120　CiscoRTPBitRate のフィールド

インターフェイス	フィールド	説明
static int	R5_3	この定数は 5.3k G.723 ビット レートです。
static int	R6_4	この定数は 6.4k G.723 ビット レートです。

メソッド

なし

関連資料

CiscoRTPInputProperties.getBitRate()、CiscoRTPOutputProperties.getBitRate() を参照してください。

CiscoRTPHandle

CiscoProvider.getCall(CiscoRTPHandle) を使用してコール参照を取得するために、CiscoRTPHandle オブジェクトを使用します。このオブジェクトは CiscoMediaCallOpenLogicalChannelEv で返されます。CiscoMediaCallOpenLogicalChannelEv event をどこで受信したかに応じて、CiscoMediaTerminal または CiscoRouteTerminal の setRTPParams パラメータでこのハンドルを渡します。

コール オブザーバが追加されていない場合や、CiscoMediaCallOpen LogicalChannelEv が送信されたときにコール オブザーバが追加されていなかった場合、CiscoProvider.getCall(CiscoRTPHandle) はメソッドとして null を返します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPHandle

フィールド

なし

メソッド

表 6-121　CiscoRTPHandle のメソッド

インターフェイス	メソッド	説明
Int	getHandle()	コールの Cisco Unified Communications Manager CallLeg ID を整数の形式で返します。

関連資料

なし

CiscoRTPInputKeyEv

CiscoRTPInputKeyEv イベント インターフェイスは、暗号化された着信メディア ストリームの鍵情報を提供します。アプリケーションは CiscoTermEvFilter.setRTPKeyEventsEnabled(true) を使用してフィルタを設定し、TerminalObserver.terminalChangedEvent() を介してこのイベントを取得する必要があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPInputKeyEv extends CiscoTermEv

フィールド

表 6-122　CiscoRTPInputKeyEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-123　CiscoRTPInputKeyEv のメソッド

インターフェイス	メソッド	説明
int	getCiscoMediaEncryptionKeyInfo()	プロバイダーが TLS リンクで開かれており、Cisco Unified Communications Manager administration でアプリケーションの SRTP を有効にするオプションが設定されている場合にだけ、CiscoMediaEncryptionKeyInfo を返します。そうでない場合は null を返します。

表 6-123　CiscoRTPInputKeyEv のメソッド

インターフェイス	メソッド	説明
int	getCiscoMediaSecurityIndicator()	次の定数の 1 つのメディアのセキュリティ インジケータを返します。 • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_AVAILABLE • CiscoMediaSecurityIndicator.MEDIA_ENCRYPT_USER_NOT_AUTHORIZED • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_UNAVAILABLE.
CiscoCallID	getCallID()	このイベントの送信時に CiscoCall がすでに存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。getCallID().getCall() は この鍵が適用されるコールを指定します。
int	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。戻り値：CiscoRTPHandle。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoRTPParams と CiscoMediaSecurityIndicator を参照してください。

CiscoRTPInputProperties

CiscoRTPInputProperties インターフェイスは、端末が受信したメディアのプロパティを返します（受信メディア ストリーム）。CiscoRTPInputStartedEv はメディアが起動していることを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPInputProperties

フィールド

なし

メソッド

表 6-124　CiscoRTPInputProperties のメソッド

インターフェイス	メソッド	説明
int	getBitRate()	CiscoRTPBitRate.R5_3 または CiscoRTPBitRate.R6_4 のメディア ビット レートを返します。
boolean	getEchoCancellation()	アプリケーションでエコー キャンセルを使用する必要がある場合は、true を返します。
java.net.InetAddress	getLocalAddress()	メディアが向けられるアドレスを返します。
int	getLocalPort()	メディアが向けられるポートを返します。
int	getPacketSize()	パケット サイズ（ミリ秒単位）を返します。

表 6-124　CiscoRTPInputProperties のメソッド （続き）

インターフェイス	メソッド	説明
int	getPayloadType()	次の定数の 1 つのペイロード形式を返します。 • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.WIDEBAND_256K

関連資料

CiscoRTPPayload と CiscoRTPBitRate を参照してください。

CiscoRTPInputStartedEv

CiscoRTPInputStartedEv イベントは着信メディアの起動を示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPInputStartedEv extends CiscoTermEv

フィールド

表 6-125　CiscoRTPInputStartedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-126　CiscoRTPInputStartedEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	CiscoCallID を返します。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。
int	getMediaConnectionMode()	CiscoMediaConnectionMode を返します。

表 6-126　CiscoRTPInputStartedEv のメソッド

インターフェイス	メソッド	説明
int	getRTPInputProperties()	着信メディアの特性を示す CiscoRTPInputProperties を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoRTPInputProperties、CiscoCallID、CiscoRTPParams、および CiscoMediaConnectionMode を参照してください。

CiscoRTPInputStoppedEv

CiscoRTPInputStoppedEv イベントは着信メディア ストリームが停止したことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPInputStoppedEv extends CiscoTermEv

フィールド

表 6-127　CiscoRTPInputStoppedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-128　CiscoRTPInputStoppedEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	CiscoCallID を返します。CiscoRTPInputStartedEv は CiscoCallID.getCall() に適用されます。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。

表 6-128　CiscoRTPInputStoppedEv のメソッド

インターフェイス	メソッド	説明
int	getMediaConnectionMode()	次のいずれかの値の mediaMode とともに CiscoMediaConnectionMode を返します。 • CiscoMediaConnectionMode.RECEIVE_ONLY（一方向メディア、受信だけ） • CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE：（双方向メディア） 通常、モード NONE のイベントを受け取ることはありません。受け取った場合は、そのイベントを無視してエラーを記録してください。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoMediaConnectionMode、CiscoCallID および CiscoRTPParams を参照してください。

CiscoRTPOutputKeyEv

CiscoRTPOutputKeyEv イベントは、暗号化された発信メディア（転送）ストリームの鍵情報を提供します。アプリケーションは CiscoTermEvFilter.setRTPKeyEventsEnabled(true) を使用してフィルタを設定し、TerminalObserver.terminalChangedEvent() を介してこのイベントを取得します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputKeyEv extends CiscoTermEv

フィールド

表 6-129　CiscoRTPOutputKeyEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-130　CisctoRTPOutputKeyEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	このイベントの送信時に CiscoCall がすでに存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。

表 6-130　CisctoRTPOutputKeyEv のメソッド （続き）

インターフェイス	メソッド	説明
CiscoMedia EncryptionKeyInfo	getCiscoMediaEncryptionKeyInfo()	プロバイダーが TLS リンクで開かれており、Cisco Unified Communications Manager administration でアプリケーションの SRTP を有効にするオプションが設定されている場合にだけ、CiscoMediaEncryptionKeyInfo を返します。そうでない場合は null を返します。
int	getCiscoMediaSecurityIndicator()	次の定数の 1 つのメディアのセキュリティ インジケータを返します。 • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_AVAILABLE • CiscoMediaSecurityIndicator.MEDIA_ENCRYPT_USER_NOT_AUTHORIZED • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_UNAVAILABLE
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoRTPParams、および CiscoMediaSecurityIndicator を参照してください。

CiscoRTPOutputProperties

CiscoRTPPutputProperties インターフェイスは、端末によって転送されたメディアのプロパティを提供します。CiscoRTPOutPutStartedEv はメディアが起動していることを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPOutputProperties

フィールド

なし

メソッド

表 6-131　CiscoRTPOutputProperties のメソッド

インターフェイス	メソッド	説明
int	getBitRate()	次の定数の 1 つのメディア ビット レートを返します。 • CiscoRTPBitRate.R5_3 • CiscoRTPBitRate.R6_4
int	getMaxFramesPerPacket()	パケットあたりの最大送信フレーム数を返します。
int	getPacketSize()	パケット サイズ（ミリ秒単位）を返します。
int	getPayloadType()	次の定数の 1 つのペイロード形式を返します。 • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE • CiscoRTPPayload.WIDEBAND_256K
int	getPrecedenceValue()	優先順位の値を返します。
java.net.InetAddress	getRemoteAddress()	メディアの送信先アドレスを返します。
int	getRemotePort()	メディアの送信先ポートを返します。
boolean	getSilenceSuppression()	Cisco Unified Communication Manager サービス パラメータ「無音抑止」が False に設定されている場合は false を返し、そうでない場合は True を返します。

関連資料

CiscoRTPBitRate を参照してください。

CiscoRTPOutputStartedEv

CiscoRTPOutputStartedEv イベント インターフェイスはメディア転送の開始を示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputStartedEv extends CiscoTermEv

フィールド

表 6-132　CiscoRTPOutputStartedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-133　CiscoRTPOutputStartedEv のメソッド

インターフェイス	メソッド	説明
CiscoRTPOutputProperties	getRTPOutputProperties()	RTP 出力プロパティを返します。
CiscoCallID	getCallID()	CiscoCallID を返します。CiscoRTPOutputStartedEv は CiscoCallID.getCall() に適用されます。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。
int	getMediaConnectionMode()	次のいずれかの値の mediaMode とともに CiscoMediaConnectionMode を返します。 • CiscoMediaConnectionMode.TRANSMIT_ONLY（一方向メディア、送信だけ） • CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE：（双方向メディア） **（注）** 通常、モード NONE のイベントを受け取ることはありません。受け取った場合は、そのイベントを無視してエラーを記録してください。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoCallID、および CiscoRTPParams を参照してください。

CiscoRTPOutputStoppedEv

CiscoRTPOutputStoppedEv イベントはメディア転送が停止したことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputStoppedEv extends CiscoTermEv

フィールド

表 6-134　CiscoRTPOutputStoppedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-135　CiscoRTPOutputStoppedEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	CiscoCallID を返します。CiscoRTPOutputStoppedEv は CiscoCallID.getCall() に適用されます。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。
int	getMediaConnectionMode()	• 次のいずれかの値で CiscoMediaConnectionMode を返します。 • CiscoMediaConnectionMode.TRANSMIT_ONLY（一方向メディア、送信） • onlyCiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE（双方向メディア） **（注）** 通常、モード NONE のイベントを受け取ることはありません。受け取った場合は、そのイベントを無視してエラーを記録してください。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoCallID、CiscoRTPParams、および CiscoMediaConnectionMode を参照してください。

CiscoRTPOutputKeyEv

CiscoRTPOutputKeyEv イベントは、暗号化された発信メディア（転送）ストリームの鍵情報を提供します。アプリケーションは CiscoTermEvFilter.setRTPKeyEventsEnabled(true) を使用してフィルタを設定し、TerminalObserver.terminalChangedEvent() を介してこのイベントを取得します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputKeyEv extends CiscoTermEv

フィールド

表 6-136　CiscoRTPOutputKeyEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-137　CisctoRTPOutputKeyEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	このイベントの送信時に CiscoCall がすでに存在している場合、CiscoCallID オブジェクトを返します。CiscoCall がない場合、このメソッドは null を返します。
CiscoMedia EncryptionKeyInfo	getCiscoMediaEncryptionKeyInfo()	プロバイダーが TLS リンクで開かれており、Cisco Unified Communications Manager administration でアプリケーションの SRTP を有効にするオプションが設定されている場合にだけ、CiscoMediaEncryptionKeyInfo を返します。そうでない場合は null を返します。
int	getCiscoMediaSecurityIndicator()	次の定数の 1 つのメディアのセキュリティ インジケータを返します。 • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_AVAILABLE • CiscoMediaSecurityIndicator.MEDIA_ENCRYPT_USER_NOT_AUTHORIZED • CiscoMediaSecurityIndicator.MEDIA_ENCRYPTED_KEYS_UNAVAILABLE
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoRTPParams、および CiscoMediaSecurityIndicator を参照してください。

CiscoRTPOutputProperties

CiscoRTPPutputProperties インターフェイスは、端末によって転送されたメディアのプロパティを提供します。CiscoRTPOutPutStartedEv はメディアが起動していることを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPOutputProperties

フィールド

なし

メソッド

表 6-138　CiscoRTPOutputProperties のメソッド

インターフェイス	メソッド	説明
int	getBitRate()	次の定数の 1 つのメディア ビット レートを返します。 • CiscoRTPBitRate.R5_3 • CiscoRTPBitRate.R6_4
int	getMaxFramesPerPacket()	パケットあたりの最大送信フレーム数を返します。
int	getPacketSize()	パケット サイズ（ミリ秒単位）を返します。
int	getPayloadType()	次の定数の 1 つのペイロード形式を返します。 • CiscoRTPPayload.NONSTANDARD • CiscoRTPPayload.G711ALAW64K • CiscoRTPPayload.G711ALAW56K • CiscoRTPPayload.G711ULAW64K • CiscoRTPPayload.G711ULAW56K • CiscoRTPPayload.G722_64K • CiscoRTPPayload.G722_56K • CiscoRTPPayload.G722_48K • CiscoRTPPayload.G7231 • CiscoRTPPayload.G728 • CiscoRTPPayload.G729 • CiscoRTPPayload.G729ANNEXA • CiscoRTPPayload.IS11172AUDIOCAP • CiscoRTPPayload.IS13818AUDIOCAP • CiscoRTPPayload.ACY_G729AASSN • CiscoRTPPayload.DATA64 • CiscoRTPPayload.DATA56 • CiscoRTPPayload.GSM • CiscoRTPPayload.ACTIVEVOICE • CiscoRTPPayload.WIDEBAND_256K
int	getPrecedenceValue()	優先順位の値を返します。
java.net.InetAddress	getRemoteAddress()	メディアの送信先アドレスを返します。
int	getRemotePort()	メディアの送信先ポートを返します。
boolean	getSilenceSuppression()	Cisco Unified Communication Manager サービス パラメータ「無音抑止」が False に設定されている場合は false を返し、そうでない場合は True を返します。

関連資料

CiscoRTPBitRate を参照してください。

CiscoRTPOutputStartedEv

CiscoRTPOutputStartedEv イベント インターフェイスはメディア転送の開始を示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputStartedEv extends CiscoTermEv

フィールド

表 6-139　CiscoRTPOutputStartedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-140　CiscoRTPOutputStartedEv のメソッド

インターフェイス	メソッド	説明
CiscoRTPOutputProperties	getRTPOutputProperties()	RTP 出力プロパティを返します。
CiscoCallID	getCallID()	CiscoCallID を返します。CiscoRTPOutputStartedEv は CiscoCallID.getCall() に適用されます。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。
int	getMediaConnectionMode()	次のいずれかの値の mediaMode とともに CiscoMediaConnectionMode を返します。 • CiscoMediaConnectionMode.TRANSMIT_ONLY（一方向メディア、送信だけ） • CiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE（双方向メディア） **（注）** 通常、モード NONE のイベントを受け取ることはありません。受け取った場合は、そのイベントを無視してエラーを記録してください。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoCallID、および CiscoRTPParams を参照してください。

CiscoRTPOutputStoppedEv

CiscoRTPOutputStoppedEv イベントはメディア転送が停止したことを示します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoRTPOutputStoppedEv extends CiscoTermEv

フィールド

表 6-141　CiscoRTPOutputStoppedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-142　CiscoRTPOutputStoppedEv のメソッド

インターフェイス	メソッド	説明
CiscoCallID	getCallID()	CiscoCallID を返します。CiscoRTPOutputStoppedEv は CiscoCallID.getCall() に適用されます。
CiscoRTPHandle	getCiscoRTPHandle()	CiscoRTPHandle オブジェクトを返します。アプリケーションは、CiscoProvider.getCall を使用してコール参照を取得できます。コール オブザーバがない場合や、このイベントの配信時にコール オブザーバがなかった場合、CiscoProvider.getCall は null を返す可能性があります。
int	getMediaConnectionMode()	• 次のいずれかの値で CiscoMediaConnectionMode を返します。 • CiscoMediaConnectionMode.TRANSMIT_ONLY（一方向メディア、送信） • onlyCiscoMediaConnectionMode.TRANSMIT_AND_RECEIVE（双方向メディア） **（注）** 通常、モード NONE のイベントを受け取ることはありません。受け取った場合は、そのイベントを無視してエラーを記録してください。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）、CiscoCallID、CiscoRTPParams、および CiscoMediaConnectionMode を参照してください。

CiscoRTPPayload

RTPPayload インターフェイスには、RTP 形式を記述する定数が入ります。CiscoRTPInputProperties.getPayloadType と CiscoRTPOutputProperties.getPayloadType メソッドはこれらの定数を返します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoRTPPayload

フィールド

表 6-143　CiscoRTPPayload のフィールド

インターフェイス	フィールド	説明
static final int	NONSTANDARD	標準外の RTP ペイロード
static final int	G711ALAW64K	G.711 64K a-law ペイロード
static final int	G711ALAW56K	G.711 56K a-law ペイロ [ド
static final int	G711ULAW64K	G.711 64K u-law ペイロード
static final int	G711ULAW56K	G.711 56K u-law ペイロード
static final int	G722_64K	G.722 64K ペイロード
static final int	G722_56K	G.722 56K ペイロード
static final int	G722_48K	G.722 48K ペイロード
static final int	G7231	G.723.1 ペイロード
static final int	G728	G.728 ペイロード
static final int	G729	G.729 ペイロード

表 6-143　CiscoRTPPayload のフィールド （続き）

インターフェイス	フィールド	説明
static final int	G729ANNEXA	G.729a ペイロード
static final int	IS11172AUDIOCAP	IS11172AUDIOCAP ペイロード
static final int	IS13818AUDIOCAP	IS13818AUDIOCAP ペイロード
static final int	ACY_G729AASSN	ACY_G729AASSN ペイロード
static final int	DATA64	DATA64 ペイロード
static final int	DATA56	DATA56 ペイロード
static final int	GSM	GSM ペイロード
static final int	ACTIVEVOICE	ACTIVEVOICE ペイロード
static final int	WIDEBAND_256K	Wide_Band_256k ペイロード

メソッド

なし

関連資料

CiscoRTPInputProperties.getPayloadType()、CiscoRTPOutputProperties.getPayloadType() および 「定数フィールド値」（P.F-1）を参照してください。

CiscoSynchronousObserver

Cisco JTAPI 実装は、アプリケーションによって Call.connect() と TerminalConnection.answer() などのブロッキング用 JTAPI メソッドを、そのオブザーバのコールバック内部から呼び出せる設計になっています。これは、オブザーバのコールバック内部から JTAPI メソッドを使用しないように警告する JTAPI 仕様の制限に、アプリケーションが制約されないことを意味します。

通常、アプリケーションが JTAPI オブジェクトに新規のオブザーバを追加すると、Cisco JTAPI 実装は、新規のオブザーバへのサービスを行うためのイベント キューとそれに付随する作業スレッドを作成します。同じオブザーバが別のオブジェクトに追加された場合、そのキューとスレッドが再利用されます。実質、オブザーバごとに、単一のキューと作業スレッドがあります。前述したとおり、この方式の利点は、アプリケーションがオブザーバ コールバック内からブロッキング用 JTAPI メソッドを呼び出すことが可能である点です。ただし、微妙な欠点は、Call.getConnections() および Connection.getState() などのアクセス用メソッドが、オブザーバ コールバック内から呼び出されたときに、イベントと一貫する結果を返さないことがある点です。

たとえば、アプリケーションが、アドレス「A」からアドレス「B」へのコールを作成し接続したとします。アプリケーションがアドレス「A」を監視している場合、CallActiveEv を受け取ったときには、そのコールの状態が当然 Call.ACTIVE になると考えられます。しかし、必ずしも真であるとは限りません。これは、アプリケーションにイベントを配送する作業スレッドが、オブジェクトの状態を更新し

ている内部 JTAPI スレッドから切り離されているためです。事実、「B」が「A」からのコールを拒絶した場合、作業スレッドが CallActiveEv を配送する時点に応じて、コール オブジェクトは、Call.ACTIVE または Call.INVALID のいずれかの状態にあります。

多くのアプリケーションは、この非同期動作の悪影響を受けることはありません。ただし、オブザーバ コールバック中にコヒーレント コール モデルの機能を使用するアプリケーションでは、Cisco JTAPI 実装のキューイング ロジックを選択的に無効にできます。対象のオブザーバ オブジェクト上に CiscoSynchronousObserver インターフェイスを実装するアプリケーションは、そのオブザーバへのイベント配送の同期を取ることを宣言します。同期オブザーバに配送されるイベントは、オブザーバ コールバックの内部から照会されるコール モデル オブジェクトの状態と一致します。

CiscoSynchronousObserver インターフェイスの実装されたオブジェクトでは、そのイベント コールバック内部からブロッキング用 JTAPI メソッドを呼び出さないでください。これを行った場合の影響は予測不能であり、JTAPI 実装が停止する可能性もあります。一方、Call.getConnections() または Connection.getState() などのすべての JTAPI オブジェクトのアクセス用メソッドは安全に使用できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoSynchronousObserver

フィールド

なし

メソッド

なし

関連資料

なし

CiscoTermActivatedEv

端末が監視され、Restriction 状態がアクティブに変更された場合、このイベントがアプリケーションに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoTermActivatedEv extends CiscoProvEv

フィールド

表 6-144　CiscoTermActivatedEv のフィールド

インターフェイス	フィールド	説明
static int	ID	なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-145　CiscoTermActivatedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Terminal	getTerminal()	有効化された端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermButtonPressedEv

CiscoTermButtonPressedEv イベントは、Terminal でボタンが押されたときに TerminalObserver に対して配信されます。アプリケーションがイベントを受信するには、ciscoTermEvFilter.setButtonPressedEnabled(true) を使用してフィルタを設定する必要があります。ボタン押下イベントは、このインターフェイスの定数（0 ～ 9、*、#、A、B、C、D など）に挙げられている数値キーパッド ボタンが Terminal で押されたときにだけ応答します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermButtonPressedEv extends CiscoTermEv

フィールド

表 6-146　CiscoTermButtonPressedEv のフィールド

インターフェイス	フィールド
static int	CHARA
static int	CHARB
static int	CHARC
static int	CHARD
static int	EIGHT
static int	FIVE
static int	FOUR
static int	ID
static int	NINE
static int	ONE
static int	POUND
static int	SEVEN
static int	SIX
static int	STAR
static int	THREE
static int	TWO
static int	ZERO

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-147　CiscoTermButtonPressedEv のメソッド

インターフェイス	メソッド	説明
int	getButtonPressed ()	端末上で押されたボタン

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoTermEvFilter と 「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnMonitoringEndEv

コールに対する監視が停止した場合、またはコールが接続解除された場合に、システムは CiscoTermConnMonitoringEndEv イベントをコール オブザーバに配信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnMonitoringEndEv extends javax.telephony.events.TermConnEv

フィールド

表 6-148　CiscoTermConnMonitoringEndEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-149　CiscoTermConnMonitoringEndEv のメソッド

インターフェイス	メソッド	説明
Int	getMonitorType()	監視のタイプを返します。返される値は常に CiscoCall.SILENT_MONITOR です。

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnMonitoringStartEv

コールに対する監視が開始されたとき、システムは CiscoTermConnMonitoringStartEv イベントをコール オブザーバに配信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnMonitoringStartEv extends javax.telephony.events.TermConnEv

フィールド

表 6-150　CiscoTermConnMonitoringStartEv のフィールド

インターフェイス	フィールド
static final int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE,

CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-151　CiscoTermConnMonitoringStartEv のメソッド

インターフェイス	メソッド	説明
int	getMonitorType()	監視のタイプを返します。返される値は常に CiscoCall.Silent_Monitor です。

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

詳細については、「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnMonitorInitiatorInfoEv

CiscoTermConnMonitorInitiatorInfoEv イベント インターフェイスは TermConnEv インターフェイスを拡張し、モニタリングのターゲット（エージェント）上の CallObserver によって報告されます。このインターフェイスは、モニタリング セッションが確立されたときに、モニタリングの開始側（スーパーバイザ）に関する情報を提供します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnMonitorInitiatorInfoEv extends javax.telephony.events.TermConnEv

フィールド

表 6-152　CiscoTermConnMonitorInitiatorInfoEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-153　CiscoTermConnMonitorInitiatorInfoEv のメソッド

インターフェイス	メソッド	説明
CiscoMonitorInitiatorInfo	getCiscoMonitorInitiatorInfo()	モニタリングの開始側の端末名とアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnMonitorTargetInfoEv

CiscoTermConnMonitorTargetInfoEv イベント インターフェイスは TermConnEv インターフェイスを拡張し、モニタリングの開始側の CallObserver によって報告されます。このインターフェイスは、モニタリング セッションが確立されたときに、モニタリング ターゲット（エージェント）に関する情報を提供します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnMonitorTargetInfoEv extends javax.telephony.events.TermConnEv

フィールド

表 6-154　CiscoTermConnMonitorTargetInfoEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY,

META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-155　CiscoTermConnMonitorTargetInfoEv のメソッド

インターフェイス	メソッド	説明
CiscoMonitorTargetInfo	getCiscoMonitorTargetInfo()	モニタリング ターゲットの端末名とアドレスを返します。

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnPrivacyChangedEv

TeminalConnection のプライバシー ステータスが変更されると、CiscoTermConnPrivacyChangedEv イベントが送信されます。プライバシーがオンの場合は、ユーザはコールに割り込みできません。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoTermConnPrivacyChangedEv

フィールド

表 6-156　CiscoTermConnPrivacyChangedEv のフィールド

インターフェイス	フィールド
static int	ID

メソッド

表 6-157　CiscoTermConnPrivacyChangedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.TerminalConnection	getTerminalConnection()	プライバシーが変更された TerminalConnection を返します。TerminalConnection で getPrivacyStatus を呼び出して、プライバシー ステータスを確認することができます。

関連資料

「定数フィールド値」（P.F-1）と CiscoTerminalConnection.getPrivacyStatus() を参照してください。

CiscoTermConnRecordingEndEv

コール録音が停止したとき、JTAPI は CiscoTermConnRecordingEndEv をコール オブザーバに配信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnRecordingEndEv extends javax.telephony.events.TermConnEv

フィールド

static intID

継承したフィールド

インターフェイス javax.telephony.events.Ev から継承したフィールド

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnRecordingStartEv

コール録音が開始されたとき、JTAPI は CiscoTermConnRecordingStartEv をコール オブザーバに配信します。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnRecordingStartEv extends javax.telephony.events.TermConnEv

フィールド

表 6-158　CiscoTermConnRecordingStartEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermConnRecordingTargetInfoEv

JTAPI は CiscoTermConnRecordingTargetInfoEv を録音元のコール オブザーバに配信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnRecordingTargetInfoEv extends javax.telephony.events.TermConnEv

フィールド

表 6-159　CiscoTermConnRecordingTargetInfoEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-160　CiscoTermConnRecordingTargetInfoEv のメソッド

インターフェイス	メソッド	説明
CiscoRecorderInfo	getCiscoRecorderInfo()	録音デバイスの端末名およびアドレスを提供する CiscoRecorderInfo を返します。

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）および CiscoRecorderInfo を参照してください。

CiscoTermConnSelectChangedEv

TerminalConnection のコール選択ステータスが変更されると、JTAPI は CiscoTermConnSelectChangedEv を、機能の呼び出しまたは手動によって送信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, javax.telephony.events.Ev, javax.telephony.events.TermConnEv

宣言

public interface CiscoTermConnSelectChangedEv extends javax.telephony.events.TermConnEv

フィールド

表 6-161　CiscoTermConnSelectChangedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.TermConnEv から
getTerminalConnection

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermCreatedEv

CiscoTerminal がプロバイダー ドメインに追加されると、JTAPI は CiscoTermCreatedEv イベントをアプリケーションのプロバイダー オブザーバに送信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoTermCreatedEv extends CiscoProvEv

フィールド

表 6-162　CiscoTermEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-163　CiscoTermCreatedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Terminal	getTerminal()	このイベントが送信された端末オブジェクトを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDataEv

電話機が XSI データ（XML オブジェクト）を受信すると、JTAPI は CiscoTermDataEv イベントを端末オブザーバに送信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDataEv extends CiscoTermEv

フィールド

表 6-164　CiscoTermDataEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-165　CiscoTermDataEv のメソッド

インターフェイス	メソッド	説明
java.lang.String	getData()	推奨されません。byte[] getTermData を使用してください。
byte[]	getTermData()	電話機が受信した XSI データに対応する XML で符号化されたバイト配列を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDeviceStateActiveEv

端末のいずれかのアドレスにあらゆる状態の発信コールがある場合、または TerminalConnection が ACTIVE 状態で CallCtlTerminalConnection が TALKING 状態の着信コールがある場合には、CiscoTermDeviceStateActiveEv イベントがアプリケーションに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDeviceStateActiveEv extends CiscoTermEv

フィールド

表 6-166　CiscoTermDeviceStateActiveEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDeviceStateAlertingEv

端末上に CallCtlConnection.Established 状態の接続で発信コールまたは着信コールがあるアドレスがなく、端末上の少なくとも 1 つのアドレスに CallCtlConnection.OFFERED 状態または CallCtlConnection.ALERTING 状態の接続で着信コールがある場合、CiscoTermDeviceStateAlertingEv イベントが 端末オブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDeviceStateAlertingEv extends CiscoTermEv

フィールド

表 6-167　CiscoTermDeviceStateAlertingEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE,

CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDeviceStateHeldEv

端末のアドレスのすべてのコールが CallCtlTerminalConnection.HELD 状態の TerminalConnection の場合は、CiscoTermDeviceStateHeldEv イベントが端末オブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDeviceStateHeldEv extends CiscoTermEv

フィールド

表 6-168　CiscoTermDeviceStateHeldEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から継承したフィールド

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から

getTerminal

インターフェイス javax.telephony.events.Ev から

getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDeviceStateIdleEv

端末のどのアドレスにもコールがない場合は、CiscoTermDeviceStateIdleEv が端末オブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDeviceStateIdleEv extends CiscoTermEv

フィールド

表 6-169　CiscoTermDeviceStateIdleEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDeviceStateWhisperEv

CiscoTermDeviceStateActiveEv イベントは、端末上の少なくとも 1 つのアドレスがインターコム ターゲットで、TerminalConnection または CallCtlTerminalConneciton の状態が Passive または Bridged であるインターコム コールが存在する場合に端末オブザーバ送信されます。この状態では、ユーザは新しい発信コールを開始したり、新しい着信コールを受けられます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDeviceStateWhisperEv extends CiscoTermEv

フィールド

表 6-170　CiscoTermDeviceStateWhisperEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermDNDOptionChangedEv

CiscoTermDNDOptionChangedEv イベントは、DND（サイレント）オプションが変更された場合に端末オブザーバに送信されます。このイベントは、アプリケーションでイベントを受信するフィルタが有効になっている場合にだけ送信されます。このイベントは、アプリケーション オブザーバで提供されます。

履歴

Cisco Unified Communications Manager リリース	説明
7.0(1)	拡張が追加されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

public interface CiscoTermDNDOptionChangedEv

extends CiscoTermEv

フィールド

表 6-171　CiscoTermDNDOptionChangedEv のフィールド

インターフェイス	フィールド
static final int	ID

表 6-172　継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

表 6-173　継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-174　CiscoTermDNDOptionChangedEv のメソッド

インターフェイス	メソッド	説明
Int	getDNDOption()	このインターフェイスは、アプリケーションに現在の DND（サイレント）オプションを返します。int dndOption を返します。

表 6-175　継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

表 6-176　継承したメソッド

インターフェイス javax.telephony.events.TermEv から
getTerminal

表 6-177　継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

「定数フィールド値」（P.F-1）と CiscoTermEv を参照してください。

CiscoTermDNDStatusChangedEv

CiscoTermDNDStatusChangedEv イベントは、DND（サイレント）のステータスが変更された場合に端末オブザーバに送信されます。ただし、アプリケーションでイベントを受信するフィルタが有効になっている必要があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermDNDStatusChangedEv extends CiscoTermEv

フィールド

表 6-178　CiscoTermDNDStatusChangedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-179　CiscoTermDNDStatusChangedEv のメソッド

インターフェイス	メソッド	説明
boolean	getDNDStatus()	アプリケーションに現在の DND（サイレント）ステータスを返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoTermEvFilter、CiscoTermEv および「定数フィールド値」（P.F-1）を参照してください。

CiscoTermEv

JTAPI のコアである javax.telephony.events.TermEv インターフェイスを拡張する CiscoTermEv インターフェイスは、Cisco によって拡張されたすべての JTAPI Terminal イベントの基本インターフェイスになります。このパッケージのコール関連イベントはすべて、直接的または間接的にこのインターフェイスを拡張します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

サブインターフェイス

CiscoMediaOpenLogicalChannelEv, CiscoRTPInputKeyEv, CiscoRTPInputStartedEv, CiscoRTPInputStoppedEv, CiscoRTPOutputKeyEv, CiscoRTPOutputStartedEv, CiscoRTPOutputStoppedEv, CiscoTermButtonPressedEv, CiscoTermDataEv, CiscoTermDeviceStateActiveEv, CiscoTermDeviceStateAlertingEv, CiscoTermDeviceStateHeldEv, CiscoTermDeviceStateIdleEv, CiscoTermDeviceStateWhisperEv, CiscoTermDNDOptionChangedEv, CiscoTermDNDStatusChangedEv, CiscoTermInServiceEv, CiscoTermOutOfServiceEv, CiscoTermRegistrationFailedEv, CiscoTermSnapshotCompletedEv, CiscoTermSnapshotEv

宣言

public interface CiscoTermEv extends CiscoEv, javax.telephony.events.TermEv

フィールド

なし

継承したフィールド

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CallEv を参照してください。

CiscoTermEvFilter

アプリケーションは、CiscoTermEvFilter インターフェイスを使用して、関係のない Terminal イベントを選択的に制限できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoTermEvFilter

フィールド

なし

メソッド

表 6-180　CiscoTermEvFilter のメソッド

インターフェイス	メソッド	説明
boolean	getDeviceDataEnabled()	端末の CiscoTermDataEv イベントの CiscoTermSnapshotCompletedEv のイベント フィルタのステータスを返します。デフォルトでは、無効に設定されています。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setDeviceDataEnabled(boolean enabled)	端末の CiscoTermDataEv イベントを有効または無効に設定します。
boolean	getButtonPressedEnabled()	端末の CiscoTermButtonPressedEv イベントの CiscoTermSnapshotCompletedEv のイベント フィルタのステータスを返します。デフォルトでは、無効に設定されています。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setButtonPressedEnabled(boolean enabled)	端末の CiscoTermButtonPressedEv イベントを有効または無効に設定します。
boolean	getRTPEventsEnabled()	端末の CiscoRTPInputStartedEv、CiscoRTPOutputStartedEv、CiscoRTPInputStoppedEv、および CiscoRTPOutputStoppedEv イベントのイベント フィルタのステータスを返します。デフォルトでは、無効に設定されています。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setRTPEventsEnabled(boolean enabled)	端末の CiscoRTPInputStartedEv、CiscoRTPOutputStartedEv、CiscoRTPInputStoppedEv、および CiscoRTPOutputStoppedEv イベントを有効または無効に設定します。

表 6-180　CiscoTermEvFilter のメソッド （続き）

インターフェイス	メソッド	説明
boolean	getSnapshotEnabled()	端末の CiscoTermSnapshotEv および CiscoTermSnapshotCompletedEv イベントのイベント フィルタのステータスを返します。無効な場合、CiscoTermSnapshotEv および CiscoTermSnapshotCompletedEv はアプリケーションに送信されません。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setSnapshotEnabled(boolean enabled)	端末の CiscoTermSnapshotEv および CiscoTermSnapshotCompletedEv を有効または無効に設定します。
boolean	getRTPKeyEventsEnabled()	端末の CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv イベントのイベント フィルタのステータスを返します。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setRTPKeyEventsEnabled(boolean enabled)	端末の CiscoRTPInputKeyEv および CiscoRTPOutputKeyEv イベントを有効または無効に設定します。
boolean	getDeviceStateActiveEvFilter()	端末の CiscoTermDeviceStateActiveEv イベントの CiscoTermSnapshotCompletedEv のイベント フィルタのステータスを返します。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
boolean	getDeviceStateHeldEvFilter()	端末の CiscoTermDeviceStateHeldEv イベントの CiscoTermSnapshotCompletedEv のイベント フィルタのステータスを返します。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
boolean	getDeviceStateAlertingEvFilter()	端末の CiscoTermDeviceStateAlerting イベントの CiscoTermSnapshotCompletedEv のイベント フィルタのステータスを返します。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
boolean	getDeviceStateIdleEvFilter()	CiscoTermDeviceStateIdleEv フィルタの状態を返します。イベント フィルタが有効な場合は true を返します。イベント フィルタが無効な場合は false を返します。
void	setDeviceStateActiveEvFilter(boolean filterValue)	端末に対して CiscoTermDeviceStateActiveEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。

表 6-180　CiscoTermEvFilter のメソッド （続き）

インターフェイス	メソッド	説明
void	setDeviceStateHeldEvFilter(boolean filterValue)	端末に対して CiscoTermDeviceStateHeldEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。
void	setDeviceStateAlertingEvFilter(boolean filter Value)	端末に対して CiscoTermDeviceStateAlertingEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。
void	setDeviceStateIdleEvFilter(boolean filterValue)	端末に対して CiscoTermDeviceStateIdleEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。
boolean	getDeviceStateWhisperEvFilter()	端末の CiscoTermDeviceStateWhisperEv フィルタのステータスを返します。
boolean	getDNDChangedEvFilter()	端末の CiscoTermDNDStatusChangedEv フィルタのステータスを返します。
void	setDNDChangedEvFilter(boolean filterValue)	端末に対して CiscoTermDNDStatusChangedEv フィルタを有効または無効にします。パラメータ：filterValue - void setDeviceStateWhisperEvFilter(boolean filterValue)。端末に対して CiscoTermDeviceStateWhisperEv フィルタを有効または無効にします。デフォルトでは、無効に設定されています。
boolean	getDNDOptionChangedEvFilter()	このインターフェイスは CiscoTermDNDOptionChangedEv フィルタのステータスを送信するために使用できます。端末の CiscoTermDNDOptionChangedEv フィルタのステータスを返します。
void	setDNDOptionChangedEvFilter(boolean filter Value)	このインターフェイスは、端末に対して CiscoTermDNDOptionChangedEv フィルタを有効または無効にするために使用します。パラメータ：filterValue

関連資料

なし

CiscoTerminal

標準の JTAPI では動的な端末登録の概念をサポートしていません。CiscoTerminal インターフェイスは、これをサポートするために、標準の端末インターフェイスを拡張します。すべての Cisco Unified Communications Manager デバイスは CiscoTerminal で表します。また、すべての CiscoTerminal について、現在 IN_SERVICE であるか OUT_OF_SERVICE であるかを照会できます。

CiscoTerminal が表す Cisco Unified Communications Manager デバイスが、たとえば IP フォンである場合、そのネットワーク接続が失われると、電話は OUT_OF_SERVICE になります。CiscoMediaTerminal などのその他のデバイスは、アプリケーションの要求に応じて登録され、IN_SERVICE または OUT_OF_SERVICE になります。

CiscoTerminal には API getIPAddressingMode() が含まれています。このインターフェイスは、CiscoTerminal に設定されている IP アドレッシング モードを返します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoObjectContainer, javax.telephony.Terminal

サブインターフェイス

CiscoMediaTerminal, CiscoRouteTerminal

宣言

public interface CiscoTerminal extends javax.telephony.Terminal, CiscoObjectContainer

フィールド

表 6-181　CiscoTerminal のフィールド

インターフェイス	フィールド	説明
static final int	OUT_OF_SERVICE	CiscoTerminal の getState() インターフェイスによって返されるこの定数フィールド値は、CiscoTerminal がアウトオブサービスであることを示します。
static final int	IN_SERVICE	CiscoTerminal の getState() インターフェイスによって返されるこの定数値は、CiscoTerminal がインサービスであることを示します。
static final int	DEVICESTATE_IDLE	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、現在、CiscoTerminal のどのアドレスにもコールがないことを示します。
static final int	DEVICESTATE_ACTIVE	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、CiscoTerminal のアドレスに少なくとも 1 つのアクティブなコールがあることを示します。

表 6-181　CiscoTerminal のフィールド （続き）

インターフェイス	フィールド	説明
static final int	DEVICESTATE_ALERTING	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、CiscoTerminal のアドレスに少なくとも 1 つがアラート状態になっているコールがあり、アクティブなコールがないことを示します。
static final int	DEVICESTATE_HELD	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、CiscoTerminal のアドレスに少なくとも 1 つのコールが保留中であり、アラート状態になっているコールやアクティブなコールがないことを示します。
static final int	DEVICESTATE_UNKNOWN	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、CiscoTerminal DeviceState が不明であることを示します。この状態は、デバイス ステート フィルタが無効な場合に返されます。
static final int	DEVICESTATE_WHISPER	CiscoTerminal の getDeviceState() インターフェイスによって返されるこの定数値は、CiscoTerminal のアドレスに少なくとも 1 つの一方向メディアでのインターコム コールコールがあり、保留中やアラート状態になっているコールやアクティブなコールがないことを示します。
static final int	UNKNOWN_ENCODING	この端末の CiscoTerminal.getSupportedEncoding () が UNKNOWN であることを示します。
static final int	NOT_APPLICABLE	この CiscoMediaTerminal または RoutePoint の CiscoTerminal.getSupportedEncoding () が NOT_APPLICABLE であることを示します。
static final int	ASCII_ENCODING	この端末の CiscoTerminal.getSupportedEncoding () が ASCII であり、この端末が ASCII_ENCODING だけをサポートしていることを示します。
static final int	UCS2UNICODE_ENCODING	この端末の CiscoTerminal.getSupportedEncoding () が UCS2UNICODE_ENCODING であることを示します。
static final int	DND_OPTION_NONE	CiscoTerminal の getDNDOption() インターフェイスによって返されるこの定数値は、設定されている DND（サイレント）オプションが None であることを示します。
static final int	DND_OPTION_RINGER_OFF	CiscoTerminal の getDNDOption() インターフェイスによって返されるこの定数値は、設定されている DND（サイレント）オプションが Ringer Off であることを示します。電話機で DND が有効になっている場合、その電話機に着信コールがあっても、呼出音が鳴りません。
static final int	DND_OPTION_CALL_REJECT	CiscoTerminal の getDNDOptions() インターフェイスによって返されるこの定数値は、設定されている DND（サイレント）オプションが Call Reject であることを示します。電話機で DND が有効になっている場合 A 共用回線を除いて、その電話機へのすべてのコールが拒否されます。

表 6-181　CiscoTerminal のフィールド （続き）

インターフェイス	フィールド	説明
static final int	IP_ADDRESSING_MODE_ UNKNOWN	これは、IPAddressing モードが不明であることを示します。
static final int	IP_ADDRESSING_MODE_IPV4	これは、IPAddressing モードが IPv4 であることを示します。
static final int	IP_ADDRESSING_MODE_IPV6	これは、IPAddressing モードが IPv6 であることを示します。
static final int	IP_ADDRESSING_MODE_IPV4 _V6	これは、IPAddressing モードがデュアル スタック（IPv4 と IPv6 の両方）であることを示します。
static final int	IP_ADDRESSING_MODE_ UNKNOWN_ANATRED	これは Cisco Unified CM の ANAT のために予約された IPAddressing 定数です。JTAPI では使用されません。

メソッド

表 6-182　CiscoTerminal のメソッド

インターフェイス	メソッド	説明
int	getRegistrationState()	**推奨されません。** このメソッドは getState() メソッドに置き換えられました。この端末の状態を返します。状態を表す定数は次のとおりです。 • CiscoTerminal.OUT_OF_SERVICE • CiscoTerminal.IN_SERVICE
int	getState()	この端末の状態を返します。状態を表す定数は次のとおりです。 • CiscoTerminal.OUT_OF_SERVICE • CiscoTerminal.IN_SERVICE
CiscoRTPInputProperties	getRTPInputProperties()	この端末の ACTIVE TerminalConnection に関連付けられた、RTP 入力ストリームに使用されるプロパティ。CiscoTerminal が CiscoTerminal.REGISTERED 状態であり、そのプロバイダーが Provider.IN_SERVICE 状態である必要があります。さらに、Terminal.getTerminalConnections () が、TerminalConnection.ACTIVE 状態にある端末接続を少なくとも 1 つ返す必要があります。 **例外** javax.telephony.InvalidStateException

表 6-182　CiscoTerminal のメソッド （続き）

インターフェイス	メソッド	説明
CiscoRTPOutputProperties	getRTPOutputProperties()	この端末の ACTIVE TerminalConnection に関連付けられた、RTP 出力ストリームに使用されるプロパティ。CiscoTerminal が CiscoTerminal.REGISTERED 状態であり、そのプロバイダーが Provider.IN_SERVICE 状態である必要があります。さらに、Terminal.getTerminalConnections () が、TerminalConnection.ACTIVE 状態にある端末接続を少なくとも 1 つ返す必要があります。 **例外** javax.telephony.InvalidStateException
java.lang.String	sendData(java.lang.String terminalData)	**推奨されません。** CiscoTerminal.sendData (byte[]) を使用します。 **例外** javax.telephony.InvalidStateException javax.telephony.MethodNotSupportedException
byte[]	sendData(byte[] terminalData)	CiscoTerminal は CiscoTerminal.IN_SERVICE 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。アプリケーションでは、このインターフェイスを使用して、バイト形式の XSI オブジェクトをプッシュできます。電話機がこのデータを受信すると、このメソッドは正常に終了します。アプリケーションは、このインターフェイスを使用して最大 2000 バイトのデータを送信します。それ以上のサイズのデータ送信要求は拒否されます。 **例外** PlatformException (The data did not get sent successfully), javax.telephony.InvalidStateException, and javax.telephony.MethodNotSupportedException
CiscoTermEvFilter	getFilter()	この端末に関連付けられたフィルタ オブジェクトを取得します。

表 6-182　CiscoTerminal のメソッド（続き）

インターフェイス	メソッド	説明
void	setFilter(CiscoTermEvFilter terminalEvFilter)	TerminalObserver に配信されるイベントのフィルタを設定します。このメソッドはコードの実行中いつでも起動できますが、通常は初期化の一部として Terminal イベントのセットアップに使用するか、CiscTermCreatedEv によって新規端末の作成が通知されたときに使用します。 • 例 1：使用方法の 1 つとして、通常は配信されないボタン押下イベントをオンにします。Terminal term = provider.getTerminal (name); if (term instanceof CiscoTerm) { CiscoTerm ciscoTerm = (CiscoTerm)term; CiscoTermEvFilter filter = ciscoTerm.getFilter (); filter.setButtonPressedEnabled (true); } term.addObserver (terminalObserver) • 例 2：別の使用方法として、アプリケーションに関係のないいくつかのイベントをオフにします。たとえば、純粋にコール制御だけを行っているアプリケーションでは、次のようにメディア（RTP）イベントをオフにできます。Terminal term = provider.getTerminal (name); if (term instanceof CiscoTerm) { CiscoTerm ciscoTerm = (CiscoTerm)term; CiscoTermEvFilter filter = ciscoTerm.getFilter (); filter.setRTPEventsEnabled (false); ciscoTerm.setFilter (filter); } term.addObserver (terminalObserver); term.getAddresses () [0].addCallObserver (callObserver) **（注）** フィルタを明示的に設定せずに CallObserver を追加すると、RTP イベントがオンになります。Cisco JTAPI Release 1.4 以前のこの動作は、現在も維持されています。setFilter が明示的に呼び出されると、フィルタの設定が有効になります。RTP イベントは、上記のようなコードでは配信されませんが、次の例では配信されます。Terminal term = provider.getTerminal (name); term.addObserver (terminalObserver); term.getAddresses () [0].addCallObserver (callObserver).

表 6-182　CiscoTerminal のメソッド （続き）

インターフェイス	メソッド	説明
javax.telephony. TerminalConnection	unPark(javax.telephony. Address UnParkAddress, java.lang.String ParkedAt)	terminalConnection を返します。CiscoTerminal は CiscoTerminal.IN_SERVICE 状態になっている必要があり、プロバイダーは Provider.IN_SERVICE 状態になっている必要があります。このメソッドには、アドレスと文字列を入力できます。 **パラメータ** • UnParkAddress：端末上のいずれかのアドレス。 • ParkedAt：指定される文字列は、コールが以前にパークされたシステム パーク ポートです。コールがパークされると、この文字列が返されます。 **例外** • javax.telephony.InvalidStateException（CiscoTerminal.getState() が IN_SERVICE ではありません） • PlatformException：その他のエラーはパークが解除されたときに発生します（たとえば、パーク解除番号が無効）。 • javax.telephony.InvalidArgumentException • javax.telephony.ResourceUnavailableException
int	getDeviceState()	• この端末の DeviceState を返します。DeviceState には端末のすべてのアドレスのコール状態が累積して反映されています。状態を表す定数は次のとおりです。 • CiscoTerminal.DEVICESTATE_ILDE • CiscoTerminal.DEVICESTATE_ACTIVE • CiscoTerminal.DEVICESTATE_ALERTING • CiscoTerminal.DEVICESTATE_HELD • CiscoTerminal.DEVICESTATE_UNKNOWN • CiscoTerminal.DEVICESTATE_WHISPER **例外** • javax.telephony.InvalidStateException：CiscoTerminal.getState() が IN_SERVICE ではありません。

表 6-182　CiscoTerminal のメソッド（続き）

インターフェイス	メソッド	説明
int	getSupportedEncoding ()	この端末でサポートされる符号化方式の種類を返します。このメソッドを使用して、端末が Unicode をサポートしているかどうかを確認します。この情報にアクセスするには、端末が CiscoTerminal.IN_SERVICE 状態になっている必要があります。supportedEncoding は次の定数のいずれかです。 • CiscoTerminal.UNKNOWN_ENCODING • CiscoTerminal.ASCII_ENCODING • CiscoTerminal.UCS2UNICODE_ENCODING • CiscoTerminal.NOT_APPLICABLE **例外** • javax.telephony.InvalidStateException
int	getLocale()	この端末がサポートしているロケールを返します。このメソッドにアクセスするには、端末が CiscoTerminal.IN_SERVICE 状態になっている必要があります。 **例外** • javax.telephony.InvalidStateException : CiscoTerminal. getState() が IN_SERVICE ではありません。
boolean	isRestricted()	この端末の制限状態を返します。端末が制限されている場合、その端末に関連付けられたすべてのアドレスも制限されます。戻り値：端末が制限されている場合は true を返します。そうでない場合は false を返します。
void	createSnapshot ()	端末における現在のアクティブ コールのセキュリティ状態を格納した CiscoTermSnapshotEv イベントを生成します。このメソッドにアクセスするには、端末が CiscoTerminal.IN_SERVICE 状態になっており、CiscoTermEvFilter.setSnapshotEnabled() が True に設定されている必要があります。 **例外** • javax.telephony.InvalidStateException : CiscoTerminal.getState() が IN_SERVICE ではありません。

表 6-182　CiscoTerminal のメソッド （続き）

インターフェイス	メソッド	説明
java.lang.String	getAltScript()	この端末がサポートしているロケール代替スクリプトを返します。空の値が戻り値は、この端末がサポートしていないか、または代替スクリプトで設定されていないことを示します。このメソッドにアクセスするには、端末が CiscoTerminal.IN_SERVICE 状態になっている必要があります。 **例外** • javax.telephony.InvalidStateException : CiscoTerminal.getState() が IN_SERVICE ではありません。
int	getProtocol()	端末のプロトコル（SCCP、SIP、またはなし）を報告し、次の定数のいずれかとしてこの端末のプロトコルを返します。 • CiscoTerminalProtocol.PROTOCOL_NONE • CiscoTerminalProtocol.PROTOCOL_SCCP • CiscoTerminalProtocol.PROTOCOL_SIP
void	setDNDStatus(boolean dndStatus)	DND（サイレント）ステータスを設定します。これにより、DND 機能が有効または無効になります。この機能は、ルート ポイントには適用されません。 **パラメータ** • dndStatus **例外** • javax.telephony.InvalidStateException : CiscoTerminal.getState() が IN_SERVICE ではありません。
boolean	getDNDStatus()	DND（サイレント）ステータスを報告し、dndStatus を返します。 **例外** • javax.telephony.InvalidStateException : CiscoTerminal.getState() が IN_SERVICE ではありません。

表 6-182　CiscoTerminal のメソッド（続き）

インターフェイス	メソッド	説明
int	getDNDOption()	DND（サイレント）オプションの値を返します。DND 機能は実際の電話機だけに適用されるため、この値は CiscoMediaTerminal または CiscoRouteTerminal にとっては重要ではありません。DND オプションを表す定数は次のとおりです。 • CiscoTerminal.DND_OPTION_NONE • CiscoTerminal.DND_OPTION_RINGER_OFF • CiscoTerminal.DND_OPTION_CALL_REJECT **例外** • javax.telephony.InvalidStateException：CiscoTerminal.getState() が IN_SERVICE ではありません。
java.lang.String	getEMLoginUsername	エクステンション モビリティ（EM）ログイン ユーザ名 EM のユーザが端末にログインしていない場合、このインターフェイスは null または空文字列を返します。 **例外** • javax.telephony.InvalidStateException：CiscoCiscoTerminal.getState() が IN_SERVICE ではない場合。

継承したフィールド

インターフェイス javax.telephony.Terminal から
addCallObserver, addObserver, getAddresses, getCallObservers, getCapabilities, getName, getObservers, getProvider, getTerminalCapabilities, getTerminalConnections, removeCallObserver, removeObserver

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から

関連資料

Terminal と CiscoMediaTerminal、「定数フィールド値」（P.F-1）および CiscoTermEvFilter を参照してください。

CiscoTerminalConnection

CiscoTerminalConnection インターフェイスは、CallControlTerminalConnection インターフェイスを拡張し、機能を追加します。アプリケーションは getReason メソッドを使用して、接続が確立された原因を取得できます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.callcontrol.CallControlTerminalConnection, CiscoObjectContainer, javax.telephony.TerminalConnection

宣言

public interface CiscoTerminalConnection extends javax.telephony.callcontrol.CallControlTerminalConnection, CiscoObjectContainer

フィールド

表 6-183　CiscoTerminalConnection のフィールド

インターフェイス	フィールド	説明
static final int	CISCO_SELECTEDNONE	コールが選択されていません。
static final int	CISCO_SELECTEDLOCAL	コールが選択されています。
static final int	CISCO_SELECTEDREMOTE	コールがその共用回線で選択されている場合、Passive TerminalConnection がこの選択状態を受け取ります。

継承したフィールド

インターフェイス javax.telephony.callcontrol.CallControlTerminalConnection から
BRIDGED, DROPPED, HELD, IDLE, INUSE, RINGING, TALKING, UNKNOWN

インターフェイス javax.telephony.TerminalConnection から
ACTIVE, PASSIVE

メソッド

表 6-184　CiscoTerminalConnection のメソッド

インターフェイス	メソッド	説明
boolean	getPrivacyStatus()	端末上のコールのプライバシー ステータスを返します。このインターフェイスは、プライバシーが有効になっている場合は true、そうでない場合は false を返します。アプリケーションの端末の実装でコールに関する情報を表示する前に、常に TerminalConnection のプライバシー ステータスを確認する必要があります。
int	getSelectStatus()	端末上のコールの選択ステータスを返します。コールのコール処理オペレーションを実行する前に、常に TerminalConnection の選択ステータスを確認する必要があります。次のいずれかになります。 • CiscoTerminalConnection.CISCO_SELECTED NONE • CiscoTerminalConnection.CISCO_SELECTED LOCAL • CiscoTerminalConnection.CISCO_SELECTED REMOTE

表 6-184　CiscoTerminalConnection のメソッド （続き）

インターフェイス	メソッド	説明
void	startRecording(int playTone Direction)	コールの録音を開始します。このメソッドが成功した場合は、システムは CiscoTermConnRecordingStartEv および CiscoTermConnRecordingTargetInfoEv をコール オブザーバに配信します。 **事前条件** • ((this.getTerminal()).getProvider()).getState() == Provider.IN_SERVICE • this.getCallControlState() == CallControlTerminalConnection.TALKING • ((CiscoProviderCapabilities)(this.getTerminal(). getProvider().getProviderCapabilities()).canRecord() == TRUE • this.getConnection().getAddress().getRecordingConfig (this.getTerminal()) == CiscoAddress.APPLICATION_CONTROLLED__ RECORDING **パラメータ** • playToneDirection：トーンを再生するかどうかを指定します。有効な値は次のとおりです。 – CiscoCall.PLAYTONE_NOLOCAL_OR_REMOTE – CiscoCall.PLAYTONE_LOCALONLY – CiscoCall.PLAYTONE_REMOTEONLY – CiscoCall.PLAYTONE_ BOTHLOCALANDREMOTE **例外** • javax.telephony.InvalidStateException：プロバイダーが「インサービス」状態でないか、または TerminalConnection が「TALKING」状態ではありません。 • javax.telephony.PrivilegeViolationException：アプリケーションに、このメソッドを起動する適切な権限がありません。 • javax.telephony.ResourceUnavailableException：このメソッドに必要な内部リソースがないことを意味します。 • javax.telephony.InvalidArgumentException：playToneDirection の値が有効ではありません。

表 6-184　CiscoTerminalConnection のメソッド （続き）

インターフェイス	メソッド	説明
CiscoRecorderInfo	getCiscoRecorderInfo()	録音側の端末名およびアドレスを公開する CiscoRecorderInfo を返します。コールが録音されていない場合は、null を返します。コール制御端末接続は、通話中状態である必要があります。
CiscoMonitorInitiatorInfo	getCiscoMonitorInitiatorInfo()	CiscoMonitorInitiatorInfo を返します。コールがモニタリングされていない場合は、null を返します。アプリケーションはこのメソッドをモニタリング ターゲットの端末接続に対して使用して、モニタリングの開始側に関する情報を取得するか、またはモニタリング セッションが存在しないと判断できます。
CiscoMonitorTargetInfo	getCiscoMonitorTargetInfo()	CiscoMonitorTargetInfo または null を返します。アプリケーションはモニタリングの開始側の端末接続に対してこのメソッドを使用して、モニタリング ターゲットに関する情報を取得できます。このメソッドは、モニタリング ターゲットの端末接続に対して呼び出された場合や、モニタリング セッションが存在しない場合には、null を返します。

継承したメソッド

インターフェイス javax.telephony.callcontrol.CallControlTerminalConnection から
getCallControlState, hold, join, leave, unhold

インターフェイス javax.telephony.TerminalConnection から
answer, getCapabilities, getConnection, getState, getTerminal, getTerminalConnectionCapabilities, getObject, setObject

インターフェイス com.cisco.jtapi.extensions.CiscoObjectContainer から

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTerminalObserver

アプリケーションは、Terminal.addObserver メソッドを使用して Terminal を監視する際に、このインターフェイスを実装して CiscoRTPInputStartedEv や CiscoRTPInputStoppedEv などの CiscoTermEv イベントを受信します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.TerminalObserver

宣言

public interface CiscoTerminalObserver extends javax.telephony.TerminalObserver

フィールド

なし

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.TerminalObserver から
terminalChangedEvent

関連資料

CiscoTermInServiceEv、CiscoTermOutOfServiceEv、CiscoRTPInputStartedEv、CiscoRTPInputStoppedEv、CiscoRTPOutputStartedEv、および CiscoRTPOutputStoppedEv を参照してください。

CiscoTerminalProtocol

CiscoTerminalProtocol イベントは、プロトコルのタイプを定義する定数のためのコンテナです。

インターフェイス履歴

Cisco Unified Communications Manager リリース	説明
3.x	拡張が追加されました。

スーパーインターフェイス

public interface CiscoTerminalProtocol

フィールド

表 6-185　CiscoTerminalProtocol のフィールド

インターフェイス	フィールド	説明
static int	PROTOCOL_NONE	CiscoTerminal の getProtocol() インターフェイスによって返されるこの定数値は、CiscoTerminal のプロトコルのタイプが不明であるか、または利用できないことを示します。
static int	PROTOCOL_SCCP	CiscoTerminal の getProtocol() インターフェイスによって返されるこの定数値は、CiscoTerminal のプロトコルのタイプが SCCP であることを示します。
static int	PROTOCOL_SIP	CiscoTerminal の getProtocol() インターフェイスによって返されるこの定数値は、CiscoTerminal のプロトコルのタイプが SIP であることを示します。

関連資料

詳細については、CiscoTerminal と「定数フィールド値」（P.F-1）を参照してください。

CiscoTermInServiceEv

CiscoTermInServiceEv イベントは、CiscoTerminal が動作可能であることを伝えるために、アプリケーションの TerminalObservers に送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermInServiceEv extends CiscoTermEv

フィールド

表 6-186　CiscoTermInServiceEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-187　CiscoTermInServiceEv のメソッド

インターフェイス	メソッド	説明
int	getSupportedEncoding ()	サポートされる符号化方式を返すことによって、端末が UNICODE に対応しているかどうかを報告します。サポートされる符号化方式の値は、次の定数のいずれかです。 • CiscoTerminal.UNKNOWN_ENCODING • CiscoTerminal.ASCII_ENCODING • CiscoTerminal.UCS2UNICODE_ENCODING • CiscoTerminal.NOT_APPLICABLE 戻り値：この端末でサポートされる符号化方式の整数値。
int	getLocale()	この端末がサポートしているロケールを返します。CiscoLocales インターフェイスで定義された int 値を返します。
boolean	getDNDStatus()	アプリケーションに現在の DND（サイレント）ステータスを返します。ブール値の dndStatus を返します。
int	getDNDOption()	アプリケーションに現在の DND（サイレント）オプションを返します。DND オプションを表す定数は次のとおりです。 • CiscoTerminal.DND_OPTION_NONE • CiscoTerminal.DND_OPTION_RINGER_OFF • CiscoTerminal.DND_OPTION_CALL_REJECT int dndOption を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）および CiscoLocales を参照してください。

CiscoTermOutOfServiceEv

CiscoTermOutOfServiceEv イベントは、CiscoTerminal がアウトオブサービスであることを伝えるために、アプリケーションの TerminalObservers に送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoOutOfServiceEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermOutOfServiceEv extends CiscoTermEv, CiscoOutOfServiceEv

フィールド

表 6-188　CiscoTermOutOfServiceEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE,

CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN, CAUSE_CALLMANAGER_FAILURE, CAUSE_CTIMANAGER_FAILURE, CAUSE_DEVICE_FAILURE, CAUSE_DEVICE_RESTRICTED, CAUSE_DEVICE_UNREGISTERED, CAUSE_LINE_RESTRICTED, CAUSE_NOCALLMANAGER_AVAILABLE, CAUSE_REHOME_TO_HIGHER_PRIORITY_CM, CAUSE_REHOMING_FAILURE

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス com.cisco.jtapi.extensions.CiscoOutOfServiceEv から

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermRegistrationFailedEv

プロバイダーで TerminalRegistration が失敗すると、アプリケーションはこのイベントを受信します。getErrorCode() が返すエラーは、問題について説明しています。このイベントを受け取った場合、アプリケーションは端末の再登録を試みる必要があります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermRegistrationFailedEv extends CiscoTermEv

フィールド

表 6-189　CiscoTermRegistrationFailedEv のフィールド

インターフェイス	フィールド	説明
static final int	ID	なし
static final int	MEDIA_CAPABILITY_MISMATCH	端末が別のメディア機能ですでに登録されているため、登録に失敗しました。同じ機能で再登録してください。
static final int	MEDIA_ALREADY_TERMINATED_NONE	端末がメディア終端タイプ none ですでに登録されているため、登録に失敗しました。メディア終端タイプ none で再登録してください。
static final int	MEDIA_ALREADY_TERMINATED_STATIC	端末が静的メディア終端ですでに登録されているため、登録に失敗しました。静的登録では、2 回目の登録ができません。端末が登録解除されるまで待機してください。
static final int	MEDIA_ALREADY_TERMINATED_DYNAMIC	端末が動的メディア終端ですでに登録されているため、登録に失敗しました。動的メディア終端で再登録してください。
static final int	OWNER_NOT_ALIVE	端末の登録中に、登録は競合状態でした。端末を登録してください。

表 6-189　CiscoTermRegistrationFailedEv のフィールド （続き）

インターフェイス	フィールド	説明
static final int	DB_INITIALIZATION_ERROR	端末の登録中にアドレスの初期化エラーが発生しました。端末を登録してください。
static final int	UNKNOWN	不明な内部理由のために登録に失敗しました。端末を再登録してください。
static final int	IP_ADDRESSING_MODE_MISMATCH	サポートされていない IP アドレッシング モードのために登録に失敗しました。正しい IP アドレッシング モードで端末の登録を試行してください。

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-190　CiscoTermRegistrationFailedEv のメソッド

インターフェイス	メソッド	説明
int	getErrorCode()	この例外のエラー コードを整数で返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermRemovedEv

CiscoTerminal がプロバイダー ドメインから削除されると、CiscoTermRemovedEv イベントがアプリケーションのプロバイダー オブザーバに送信されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoTermRemovedEv extends CiscoProvEv

フィールド

表 6-191　CiscoTermRemovedEv のフィールド

インターフェイス	フィールド
static final int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-192　CiscoTermRemovedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Terminal	getTerminal()	プロバイダー ドメインから削除された端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermRestrictedEv

アプリケーションの実行開始後に、ユーザが Cisco Unified Communications Manager Administration から端末を制限すると、アプリケーションは CiscoTermRestrictedEv イベントを受信します。アプリケーションは、制限された端末やその端末のアドレスに関するイベントは受信できません。インサービス状態にある端末が制限された場合、アプリケーションはこのイベントを受信し、端末およびそれに対応するアドレスはアウトオブサービス状態になります。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoProvEv, javax.telephony.events.Ev, javax.telephony.events.ProvEv

宣言

public interface CiscoTermRestrictedEv extends CiscoProvEv

フィールド

表 6-193　CiscoTermRestrictedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-194　CiscoTermRestrictedEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Terminal	getTerminal	制限された端末を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.ProvEv から
getProvider

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTermSnapshotCompletedEv

2 つのエンド ポイント間でコールが確立された後にアプリケーションが起動された場合、コール中のモニタリングに関して、アプリケーションは Terminal.createSnapshot() を照会する必要があります。端末上のすべてのアドレスのコール イベントが配信された後で、アプリケーションは CiscoTermSnapshotCompletedEv を取得します。下位互換性を維持するため、これらのイベントはアプリケーションで CiscoTermEvFilter の snapShotRTPEnabled フィルタが有効にされた場合にのみ生成されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermSnapshotCompletedEv extends CiscoTermEv

フィールド

表 6-195　CiscoTermSnapshotCompletedEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

なし

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoTermEvFilter と 「定数フィールド値」（P.F-1）を参照してください。

CiscoTermSnapshotEv

2 つのエンド ポイント間でコールが確立された後にアプリケーションが起動された場合、コール中のモニタリングに関して、アプリケーションは Terminal.createSnapshot() を照会する必要があります。スナップショット イベント CiscoTermSnapshotEv が送信され、エンドポイント間の現在のメディアがセキュアかどうかが示されます。また、アプリケーションは CiscoMediaCallSecurityIndicator を照会してコールのセキュリティ インジケータを取得することもできますが、このイベントに鍵情報が含まれません。端末のどの回線にもコールが存在しない場合、アプリケーションは CiscoTermSnapshotCompletedEv のみを取得します。下位互換性を維持するため、これらのイベントはアプリケーションで CiscoTermEvFilter の snapShotRTPEnabled フィルタが有効にされた場合にのみ生成されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

CiscoEv, CiscoTermEv, javax.telephony.events.Ev, javax.telephony.events.TermEv

宣言

public interface CiscoTermSnapshotEv extends CiscoTermEv

フィールド

表 6-196　CiscoTermSnapshotEv のフィールド

インターフェイス	フィールド
static int	ID

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-197　CiscoTermSnapshotEv のメソッド

インターフェイス	メソッド	説明
CiscoMediaCallSecurity Indicator[]	getCiscoMediaCallSecurityIndicator()	このデバイス上の各アクティブ コールにおけるメディアのセキュリティ状態を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.TermEv から
getTerminal

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

CiscoTermEvFilter と 「定数フィールド値」（P.F-1）を参照してください。

CiscoTone

CiscoTone インターフェイスは CTI Tone 定数コードを定義します。CiscoToneChangedEv は、これらの定数のいずれかを返す getTone() メソッドを提供します。ZIPZIP トーン タイプがアプリケーションに公開されます。

履歴

Cisco Unified Communications Manager リリース	説明
7.0(1)	拡張が追加されました。

スーパーインターフェイス

public interface CiscoTone

フィールド

表 6-198　CiscoTone のフィールド

インターフェイス	フィールド	説明
Static int	ZIPZIP	このインターフェイスは ZIPZIP tone の整数値を定義します。CiscoToneChangedEv.getTone() インターフェイスはトーンの整数値を返します。

CiscoToneChangedEv、「定数フィールド値」（P.F-1）も参照してください。

CiscoToneChangedEv

CiscoToneChangedEv イベントは、このコールにトーンが生成されたことを示します。このイベントは CallControlCallObserver インターフェイスによって報告されます。現在、このトーンは Forced Authorization Code （FAC）機能または Client Matter Code（CMC）機能だけによって生成されます。FAC_CMC によってトーンが作成される場合、CiscoToneChangedEv.getCiscoCause() は CiscoCallEv.CAUSE_FAC_CMC_FEATURE を返します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoToneChangedEv extends CiscoCallEv

フィールド

表 6-199　CiscoToneChangedEv のフィールド

インターフェイス	フィールド	説明
static final int	ID	なし
static final int	FAC_REQUIRED	Connection.addToAddress(String) を使用して FAC を入力する必要があることを示しています。これは FAC_CMC_FEATURE_ID だけに適用されます。
static final int	CMC_REQUIRED	Connection.addToAddress(String) を使用して CMC を入力する必要があることを示しています。これは FAC_CMC_FEATURE_ID だけに適用されます。
static final int	FAC_CMC_REQUIRED	Connection.addToAddress(String) を使用して FAC と CMC の両方を入力する必要があることを示しています。アプリケーションは、一度に 1 つずつ文字列コードを入力することも、同時に両方を入力して #（シャープ記号）文字で区切ることもできます。これは FAC_CMC_FEATURE_ID だけに適用されます。

CAUSE_ACCESSINFORMATIONDISCARDED

, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED, CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE, CAUSE_CTICCMSIP411LENGTHREQUIRED, CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG, CAUSE_CTICCMSIP414REQUESTURITOOLONG,

CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,
CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL,
CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS,
CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER,
CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR,
CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING,
CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION,
CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING,
CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL,
CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER,
CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR,
CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR,
CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL,
CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED,
CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY,
CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED,
CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT,
CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION,
CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

継承したフィールド

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から
CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から

メソッド

表 6-200　CiscoToneChangedEv のメソッド

インターフェイス	メソッド	説明
int	getTone()	CiscoTone から生成されたトーン タイプを返します。
int	getWhichCodeRequired()	ダイヤルされた DN にどのコードが必要かを返します。 odeRequired は次のいずれかです。 • CiscoToneChangedEv.FAC_REQUIRED • CiscoToneChangedEv.CMC_REQUIRED • CiscoToneChangedEv.FAC_CMC_REQUIRED

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CiscoTransferEndEv

CiscoTransferEndEv イベントは、転送動作が完了したことを示します。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoTransferEndEv extends CiscoCallEv

フィールド

なし

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE, CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL, CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE, CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT, CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE, CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED, CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN, CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED, CAUSE_CTICCMSIP406NOTACCEPTABLE, CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,

CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,
CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC,
CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL,
CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE,
CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED,
CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED,
CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS,
CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT,
CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING,
CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS,
CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER,
CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR,
CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING,
CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION,
CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING,
CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL,
CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER,
CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR,
CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR,
CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED,
CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL,
CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED,
CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY,
CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED,

CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER, CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-201　CiscoTransferEndEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Call	getTransferredCall()	転送を実行したコールを返します。このコールは Call.INVALID の状態です。
javax.telephony.Call	getFinalCall()	転送の終了後にアクティブになっているコールを返します。
javax.telephony.TerminalConnection	getTransferController()	現在、最後のコールの転送コントローラとして機能している ACTIVE TerminalConnection を返します。transferController が SharedLine の場合、複数の TerminalConnection オブジェクトがあります。このメソッドは ACTIVE TerminalConnection を返します。ただし、アプリケーションが ACTIVE TerminalConnection を監視していない場合、このメソッドは PASSIVE TerminalConnection オブジェクトのうち 1 つを返します。
javax.telephony.TerminalConnection[]	getTransferControllers()	現在、最後のコールの転送コントローラとして機能している TerminalConnection オブジェクトのリストを返します。transferController が SharedLine ではないときは、1 つの TerminalConnection だけがリストに含まれます。転送コントローラにオブザーバがない場合、このメソッドは null を返します。

表 6-201　CiscoTransferEndEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Address	getTransferControllerAddress()	現在、最後のコールの転送コントローラとして機能しているアドレスを返します。
boolean	isSuccess()	転送が正常に実行された場合は true、そうでない場合は false を返します。

継承したメソッド

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

関連資料

「定数フィールド値」（P.F-1）および getTransferControllers() を参照してください。

CiscoTransferStartEv

CiscoTransferStartEv イベントは、転送操作が開始したことを示します。このイベントは CallControlCallObserver インターフェイスによって報告されます。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

スーパーインターフェイス

javax.telephony.events.CallEv, CiscoCallEv, CiscoEv, javax.telephony.events.Ev

宣言

public interface CiscoTransferStartEv extends CiscoCallEv

フィールド

表 6-202　CiscoTransferStartEv のフィールド

インターフェイス	フィールド
static final int	ID

継承したフィールド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
CAUSE_ACCESSINFORMATIONDISCARDED, CAUSE_BARGE,
CAUSE_BCBPRESENTLYAVAIL, CAUSE_BCNAUTHORIZED, CAUSE_BEARERCAPNIMPL,
CAUSE_CALLBEINGDELIVERED, CAUSE_CALLIDINUSE,
CAUSE_CALLMANAGER_FAILURE, CAUSE_CALLREJECTED, CAUSE_CALLSPLIT,
CAUSE_CHANTYPENIMPL, CAUSE_CHANUNACCEPTABLE,
CAUSE_CTICCMSIP400BADREQUEST, CAUSE_CTICCMSIP401UNAUTHORIZED,
CAUSE_CTICCMSIP402PAYMENTREQUIRED, CAUSE_CTICCMSIP403FORBIDDEN,
CAUSE_CTICCMSIP404NOTFOUND, CAUSE_CTICCMSIP405METHODNOTALLOWED,
CAUSE_CTICCMSIP406NOTACCEPTABLE,
CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED,
CAUSE_CTICCMSIP408REQUESTTIMEOUT, CAUSE_CTICCMSIP410GONE,
CAUSE_CTICCMSIP411LENGTHREQUIRED,
CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG,
CAUSE_CTICCMSIP414REQUESTURITOOLONG,
CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE,
CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME,
CAUSE_CTICCMSIP420BADEXTENSION, CAUSE_CTICCMSIP421EXTENSTIONREQUIRED,
CAUSE_CTICCMSIP423INTERVALTOOBRIEF,
CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE,
CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST, CAUSE_CTICCMSIP482LOOPDETECTED,
CAUSE_CTICCMSIP483TOOMANYHOOPS, CAUSE_CTICCMSIP484ADDRESSINCOMPLETE,
CAUSE_CTICCMSIP485AMBIGUOUS, CAUSE_CTICCMSIP486BUSYHERE,
CAUSE_CTICCMSIP487REQUESTTERMINATED,
CAUSE_CTICCMSIP488NOTACCEPTABLEHERE, CAUSE_CTICCMSIP491REQUESTPENDING,
CAUSE_CTICCMSIP493UNDECIPHERABLE,
CAUSE_CTICCMSIP500SERVERINTERNALERROR,
CAUSE_CTICCMSIP501NOTIMPLEMENTED, CAUSE_CTICCMSIP502BADGATEWAY,
CAUSE_CTICCMSIP503SERVICEUNAVAILABLE, CAUSE_CTICCMSIP504SERVERTIMEOUT,
CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED,
CAUSE_CTICCMSIP513MESSAGETOOLARGE, CAUSE_CTICCMSIP600BUSYEVERYWHERE,
CAUSE_CTICCMSIP603DECLINE, CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE,
CAUSE_CTICCMSIP606NOTACCEPTABLE, CAUSE_CTICONFERENCEFULL,
CAUSE_CTIDEVICENOTPREEMPTABLE, CAUSE_CTIDROPCONFEREE,
CAUSE_CTIMANAGER_FAILURE, CAUSE_CTIPRECEDENCECALLBLOCKED,
CAUSE_CTIPRECEDENCELEVELEXCEEDED,
CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH, CAUSE_CTIPREEMPTFORREUSE,
CAUSE_CTIPREEMPTNOREUSE, CAUSE_DESTINATIONOUTOFORDER,
CAUSE_DESTNUMMISSANDDCNOTSUB, CAUSE_DPARK, CAUSE_DPARK_REMINDER,

CAUSE_DPARK_UNPARK, CAUSE_EXCHANGEROUTINGERROR, CAUSE_FAC_CMC, CAUSE_FACILITYREJECTED, CAUSE_IDENTIFIEDCHANDOESNOTEXIST, CAUSE_IENIMPL, CAUSE_INBOUNDBLINDTRANSFER, CAUSE_INBOUNDCONFERENCE, CAUSE_INBOUNDTRANSFER, CAUSE_INCOMINGCALLBARRED, CAUSE_INCOMPATABLEDDESTINATION, CAUSE_INTERWORKINGUNSPECIFIED, CAUSE_INVALIDCALLREFVALUE, CAUSE_INVALIDIECONTENTS, CAUSE_INVALIDMESSAGEUNSPECIFIED, CAUSE_INVALIDNUMBERFORMAT, CAUSE_INVALIDTRANSITNETSEL, CAUSE_MANDATORYIEMISSING, CAUSE_MSGNCOMPATABLEWCS, CAUSE_MSGTYPENCOMPATWCS, CAUSE_MSGTYPENIMPL, CAUSE_NETOUTOFORDER, CAUSE_NOANSWERFROMUSER, CAUSE_NOCALLSUSPENDED, CAUSE_NOCIRCAVAIL, CAUSE_NOERROR, CAUSE_NONSELECTEDUSERCLEARING, CAUSE_NORMALCALLCLEARING, CAUSE_NORMALUNSPECIFIED, CAUSE_NOROUTETODDESTINATION, CAUSE_NOROUTETOTRANSITNET, CAUSE_NOUSERRESPONDING, CAUSE_NUMBERCHANGED, CAUSE_ONLYRDIVEARERCAPAVAIL, CAUSE_OUTBOUNDCONFERENCE, CAUSE_OUTBOUNDTRANSFER, CAUSE_OUTOFBANDWIDTH, CAUSE_PROTOCOLERRORUNSPECIFIED, CAUSE_QSIG_PR, CAUSE_QUALOFSERVNAVAIL, CAUSE_QUIET_CLEAR, CAUSE_RECOVERYONTIMEREXPIRY, CAUSE_REDIRECTED, CAUSE_REQCALLIDHASBEENCLEARED, CAUSE_REQCIRCNAVIL, CAUSE_REQFACILITYNIMPL, CAUSE_REQFACILITYNOTSUBSCRIBED, CAUSE_RESOURCESNAVAIL, CAUSE_RESPONSETOSTATUSENQUIRY, CAUSE_SERVNOTAVAILUNSPECIFIED, CAUSE_SERVOPERATIONVIOLATED, CAUSE_SERVOROPTNAVAILORIMPL, CAUSE_SUBSCRIBERABSENT, CAUSE_SUSPCALLBUTNOTTHISONE, CAUSE_SWITCHINGEQUIPMENTCONGESTION, CAUSE_TEMPORARYFAILURE, CAUSE_UNALLOCATEDNUMBER,CAUSE_USERBUSY

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

インターフェイス javax.telephony.events.Ev から

CAUSE_CALL_CANCELLED, CAUSE_DEST_NOT_OBTAINABLE, CAUSE_INCOMPATIBLE_DESTINATION, CAUSE_LOCKOUT, CAUSE_NETWORK_CONGESTION, CAUSE_NETWORK_NOT_OBTAINABLE, CAUSE_NEW_CALL, CAUSE_NORMAL, CAUSE_RESOURCES_NOT_AVAILABLE, CAUSE_SNAPSHOT, CAUSE_UNKNOWN, META_CALL_ADDITIONAL_PARTY, META_CALL_ENDING, META_CALL_MERGING, META_CALL_PROGRESS, META_CALL_REMOVING_PARTY, META_CALL_STARTING, META_CALL_TRANSFERRING, META_SNAPSHOT, META_UNKNOWN

メソッド

表 6-203　CiscoTransferStartEv のメソッド

インターフェイス	メソッド	説明
javax.telephony.Call	getTransferredCall()	転送の予定されているコールを返します。
javax.telephony.Call	getFinalCall()	転送の終了後にアクティブになっているコールを返します。
javax.telephony.TerminalConnection	getTransferController()	現在、最後のコールの転送コントローラとして機能している ACTIVE TerminalConnection を返します。transferController が SharedLine の場合、複数の TerminalConnection オブジェクトがあります。このメソッドは ACTIVE TerminalConnection を返します。ただし、アプリケーションが ACTIVE TerminalConnection を監視していない場合、このメソッドは PASSIVE TerminalConnection オブジェクトのうち 1 つを返します。
javax.telephony.TerminalConnection[]	getTransferControllers()	現在、最後のコールの転送コントローラとして機能している TerminalConnection オブジェクトのリストを返します。transferController が SharedLine ではないときは、TerminalConnection だけがリストに含まれます。転送コントローラにオブザーバがない場合、このメソッドは null を返します。
javax.telephony.Address	getTransferControllerAddress()	現在、最後のコールの転送コントローラとして機能しているアドレスを返します。
String	getControllerTerminalName()	転送が行われるコントローラの端末名を返します。

継承したメソッド

インターフェイス com.cisco.jtapi.extensions.CiscoCallEv から
getCiscoCause, getCiscoFeatureReason

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

インターフェイス javax.telephony.events.CallEv から
getCall

インターフェイス javax.telephony.events.Ev から
getCause, getID, getMetaCode, getObserved, isNewMetaEvent

関連資料

「定数フィールド値」（P.F-1）および getTransferControllers() を参照してください。

CiscoUrlInfo

CiscoUrlInfo オブジェクトは、SIP エンドポイントに関連付けられた Uniform Resources Locator（URL）のプロパティを指定します。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(1 および 2)	変更を記録するために履歴表が作成されました。

宣言

public interface CiscoUrlInfo

フィールド

表 6-204　CiscoUrlInfo のフィールド

インターフェイス	フィールド	説明
static final int	TRANSPORT_TYPE_UDP	エンドポイントは UDP を使用しています。
static final int	TRANSPORT_TYPE_TCP	エンドポイントは TCP を使用しています。
static final int	URL_TYPE_UNKNOWN	不明なタイプの URL です。
static final int	URL_TYPE_TEL	テレフォニー タイプの URL です。
static final int	URL_TYPE_SIP	SIP タイプの URL です。

メソッド

表 6-205　CiscoUrlInfro のメソッド

インターフェイス	メソッド	説明
java.lang.String	getUser()	SIP エンドポイントに関連付けられたユーザ名を文字列で返します。
java.lang.String	getHost()	SIP エンドポイントに関連付けられたホスト名を返します。
int	getPort()	SIP エンドポイントに関連付けられたポートを返します。

表 6-205　CiscoUrlInfro のメソッド

インターフェイス	メソッド	説明
int	getTransportType()	SIP エンドポイントで使用されているトランスポート レイヤ プロトコルのタイプを返します。タイプは CiscoUrlInfo.TRANSPORT_TYPE_UDP または CiscoUrlInfo.TRANSPORT_TYPE_TCP のいずれかです。
int	getUrlType()	このメソッドは、エンドポイント URL タイプを返します。CiscoUrlInfo.URL_TYPE_UNKNOWN, CiscoUrlInfo.URL_TYPE_TEL、および CiscoUrlInfo.URL_TYPE_SIP が可能性のある戻り値です。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

ComponentUpdater

指定されたディレクトリにアップデータのログを作成する、オーバーロードされたメソッドが導入されました。

インターフェイス履歴

Cisco Unified Communications Manager リリース番号	説明
7.1(2)	7.1（2）で追加されました。

宣言

public interface ComponentUpdater

メソッド

指定されたディレクトリにアップデータのログを作成する、オーバーロードされたメソッドが導入されました。

表 6-206　ComponentUpdater のメソッド

インターフェイス	メソッド	説明
ComponentUpdater	String tracePath	コンサルト オペレーションがキャンセルされるコンサルト コールを返します。コンサルト コールが存在しない場合、NULL を返します。

関連資料

「定数フィールド値」（P.F-1）を参照してください。

CHAPTER 7

Cisco Unified JTAPI のアラームとサービス

Cisco Unified JTAPI のアラームとサービスは、JTAPI 1.2 の仕様ではまだ公開されていないが Cisco Unified Communications Manager では使用可能な追加機能を公開するクラスとインターフェイスのセットで構成されます。開発者はこのクラスやインターフェイスを使用して新しいアプリケーションを開発したり、既存のクラスやインターフェイスを変更して新しいメソッドを作成できます。

この章では、Cisco Unified Communications Manager の実装で使用可能なアラームとサービスについて説明します。次のような構成になっています。

Cisco Unified JTAPI 拡張の詳細については、第 6 章「Cisco Unified JTAPI 拡張」を参照してください。

アラーム クラスの階層

次のクラス階層は、com.cisco.services.alarm パッケージに含まれています。

java.lang.Object

- com.cisco.services.alarm.AlarmManager
- com.cisco.services.alarm.DefaultAlarm（com.cisco.services.alarm.Alarm を実装）
- com.cisco.services.alarm.DefaultAlarmWriter（com.cisco.services.alarm.AlarmWriter を実装）
- com.cisco.services.alarm.ParameterList

AlarmManager

AlarmManager は、Alarm オブジェクトの作成に使用します。AlarmManager は、ファシリティ名および AlarmService ホスト名とポートとともに作成されます。ファクトリにより作成されるアラームはすべて、このファシリティ名に関連付けられます。このクラスでは、システム全体で使用できる、単一の AlarmWriter への参照が維持されます。アプリケーションで、この AlarmWriter を利用できます。AlarmManager には、AlarmWriter のデフォルトの実装が用意されています。ユーザ定義の AlarmWriter を実装して、この AlarmWriter をオーバーライドすることが可能です。

使用法：
AlarmManager AlarmManager = new AlarmManager(facilityName, alarmServiceHost, alarmServicePort, debugTrace, errorTrace);

Alarm は、alarmName (mnemonic)、サブファシリティ、および重大度を指定することで、ファクトリによって作成されます。Alarm は、アプリケーションのさまざまな部分での使用に備えてキャッシュしておくことができます。アラームの送信の際、アプリケーションでは、AlarmService に特定の情報を提供する変数パラメータを指定できます。

使用法：
一般に、アプリケーションは、それ自体の AlarmManager インスタンスを維持しています。また、アプリケーションでは、アラーム トレースが既存のトレース宛先にも送信されるように、デバッグおよびエラー トレースを設定する必要があります。

Manager および Writer クラスの設定：
AlarmWriter alarmWriter = new DefaultAlarmWriter(port, alarmServiceHost);
AlarmManager alarmManager = new AlarmManager(“AA_IVR“, alarmWriter, debugTrace, errorTrace);

アラームの生成：
サブファシリティとデフォルトの重大度でアラームを作成します。
Alarm alarm = alarmManager.createAlarm(“HTTPSS“, Alarm.INFORMATIONAL);
alarm.send(“090T“) は、ニーモニック付きでアラームを送信します。
alarm.send(“090T“, “Port is stuck“, “CTIPort01“) またはニーモニックおよびパラメータ付き。

宣言

```
public class AlarmManager

java.lang.Object
      |
      +--com.cisco.services.alarm.AlarmManager
```

（注） 複数のパラメータの送信は、ParameterList を指定すれば可能です。

メンバの概要	
コンストラクタ	
	AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace) ファシリティに対して AlarmManager のインスタンスを作成します。
メソッド	
Alarm	createAlarm(String, int) subFacility に要求された重大度の Alarm を作成します。
AlarmWriter	getAlarmWriter()
void	setAlarmWriter(AlarmWriter) アプリケーションが、この AlarmManager によって使用される AlarmWriter を、ユーザ定義の AlarmWriter でオーバーライドすることを可能にします。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

AlarmManager(String, AlarmWriter, Trace, UnconditionalTrace)

```
public AlarmManager(java.lang.String facility,
    com.cisco.services.alarm.AlarmWriter writer,
    com.cisco.services.tracing.Trace debugTrace_,
    com.cisco.services.tracing.UnconditionalTrace errorTrace_)
```

ファシリティに対して AlarmManager のインスタンスを作成します。アプリケーションで、AlarmService に Alarm を送信するための、この AlarmManager により使用される AlarmWriter を指定します。

メソッド

createAlarm(String, int)

```
public com.cisco.services.alarm.Alarm createAlarm(java.lang.String
    subfacility, int severity)
```

subFacility に要求された重大度の Alarm を作成します。

戻り値：

アラーム インターフェイスを実装するオブジェクト。

getAlarmWriter()

```
public com.cisco.services.alarm.AlarmWriter getAlarmWriter()
```

戻り値：

AlarmWriter オブジェクト。

setAlarmWriter(AlarmWriter)

```
public void setAlarmWriter(com.cisco.services.alarm.AlarmWriter
    writer)
```

アプリケーションが、この AlarmManager によって使用される AlarmWriter を、ユーザ定義の AlarmWriter でオーバーライドすることを可能にします。

AlarmWriter

AlarmWriter は、アラーム メッセージを受け取り、TCP リンク上にある受信側の AlarmService にそのメッセージを転送します。このインターフェイスは、com.cisco.service.alarm の実装で使用する他の AlarmWriter を実装するために使用できます。DefaultAlarmWriter は、この実装に提供されていて、AlarmManager から取得できます。

宣言

```
public interface AlarmWriter
```

既知の実装クラスの一覧

DefaultAlarmWriter

メンバの概要

メンバの概要	
メソッド	
void	close() AlarmWriter をクローズします。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
void	send(String) アラーム メッセージを AlarmService に送信します。
void	setEnabled(boolean) AlarmWriter を有効または無効にします。

メソッド

close()

```
public void close()
```

AlarmWriter をクローズします。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

AlarmWriter の説明。

getEnabled()

```
public boolean getEnabled()
```

戻り値：

AlarmWriter の現在の有効または無効の状態。

getName()

```
public java.lang.String getName()
```

戻り値：

AlarmWriter の名前。

send(String)

```
public void send(java.lang.String alarmMessage)
```

アラーム メッセージを AlarmService に送信します。

パラメータ：

the：送信する Alarm。

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

AlarmWriter を有効または無効にします。

パラメータ：

enable：AlarmWriter を有効または無効にします。

DefaultAlarm

Alarm インターフェイスの実装。AlarmManager は、createAlarm() メソッドが呼び出されると、これらの Alarm を作成します。

宣言

```
public class DefaultAlarm implements Alarm

java.lang.Object
      |
      +--com.cisco.services.alarm.DefaultAlarm
```

実装インターフェイスの一覧

Alarm

メンバの概要

メンバの概要	
コンストラクタ	
	DefaultAlarm(String, String, int, AlarmWriter))
メソッド	
java.lang.String	getFacility()
int	getSeverity()
java.lang.String	getSubFacility()
void	send(String) 指定されたニーモニック付きでアラームを送信します。

メンバの概要 （続き）	
void	send(String, ParameterList) 指定された名前とパラメータ リスト付きでアラームを送信します。
void	send(String, String, String) 指定された名前とパラメータ付きでアラームを送信します。

継承メンバの概要

インターフェイス Alarm から継承したフィールド

ALERTS, CRITICAL, DEBUGGING, EMERGENCIES, ERROR, HIGHEST_LEVEL, INFORMATIONAL, LOWEST_LEVEL, NOTIFICATION, NO_SEVERITY, UNKNOWN_MNEMONIC, WARNING

クラス Object から継承したメソッド

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

DefaultAlarm(String, String, int, AlarmWriter)

```
public DefaultAlarm(java.lang.String facility,
    java.lang.String subFacility, int severity,
    com.cisco.services.alarm.AlarmWriter alarmWriter)
```

メソッド

getFacility()

```
public java.lang.String getFacility()
```

定義：

インターフェイス Alarm の getFacility

getSeverity()

```
public int getSeverity()
```

定義：

インターフェイス Alarm の getSeverity

getSubFacility()

```
public java.lang.String getSubFacility()
```

定義：

インターフェイス Alarm の getSubFacility

send(String)

```
public void send(java.lang.String mnemonic)
```

指定されたニーモニック付きでアラームを送信します。

定義：

インターフェイス Alarm の send

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
    com.cisco.services.alarm.ParameterList paramList)
```

指定された名前とパラメータ リスト付きでアラームを送信します。

定義：

インターフェイス Alarm の send

send(String, String, String)

```
public void send(java.lang.String mnemonic,
    java.lang.String paramName, java.lang.String paramValue)
```

指定された名前とパラメータ付きでアラームを送信します。

定義：

インターフェイス Alarm の send

DefaultAlarmWriter

AlarmWriter インターフェイスの DefaultAlarmWriter 実装。
DefaultAlarmWriter は、アラームが書き込まれる、固定サイズのキューを保持しています。アラーム サービスへのアラームの送信は、別のスレッド上で実行されます。このキューは、固定サイズです。

宣言

```
public class DefaultAlarmWriter implements AlarmWriter

java.lang.Object
      |
      +--com.cisco.services.alarm.DefaultAlarmWriter
```

実装インターフェイスの一覧

AlarmWriter

メンバの概要

メンバの概要	
コンストラクタ	
	DefaultAlarmWriter(int, String) AlarmService のホスト名、ポートを指定できる DefaultAlarmWriter のコンストラクタ。キューサイズは 50 に初期化されます。

メンバの概要 （続き）	
	DefaultAlarmWriter(int, String, int) AlarmService のホスト名、ポートおよびキュー サイズを指定できる、DefaultAlarmWriter のコンストラクタ。
	DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace) AlarmService のホスト名、ポート、キュー サイズ、デバッグトレースレベル、エラートレースレベルを指定できる、DefaultAlarmWriter のコンストラクタ。
メソッド	
void	close() スレッドをシャットダウンしてソケットを閉じます。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
static void	main(String[])
void	send(String) Alarm をアラーム サービスに送信します。
void	setEnabled(boolean) アプリケーションは、AlarmWriter を動的に有効または無効にできます。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

コンストラクタ

DefaultAlarmWriter(int, String)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName)
    throws UnknownHostException
```

AlarmService のホスト名、ポートを指定できる、DefaultAlarmWriter のコンストラクタ。キューサイズは 50 になります。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

port：アラーム サービスがアラームを受信するポート。

alarmServiceName：アラーム サービスを所有するマシンのホスト名。

例外：

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName, int queueSize)
    throws UnknownHostException
```

AlarmService のホスト名、ポート、キュー サイズ、デバッグ トレース レベル、エラー トレース レベルを指定できる、DefaultAlarmWriter のコンストラクタ。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

port：アラーム サービスがアラームを受信するポート。

alarmServiceName：アラーム サービスを所有するマシンのホスト名。

queueSize：アラーム ライターで保持されるキューのサイズ。

debugTrace：デバッグ トレース レベル。

errorTrace：エラー トレース レベル。

例外：

java.net.UnknownHostException

DefaultAlarmWriter(int, String, int, ConditionalTrace, UnconditionalTrace)

```
public DefaultAlarmWriter(int port,
    java.lang.String alarmServiceName, int queueSize,
    com.cisco.services.tracing.ConditionalTrace debugTrace_,
    com.cisco.services.tracing.UnconditionalTrace errorTrace_)
    throws UnknownHostException
```

AlarmService のホスト名、ポートおよびキュー サイズを指定できる、DefaultAlarmWriter のコンストラクタ。AlarmService は、このポートで Alarm メッセージを受信します。

パラメータ：

port：アラーム サービスがアラームを受信するポート。

alarmServiceName：アラーム サービスを所有するマシンのホスト名。

queueSize：アラーム ライターで保持されるキューのサイズ。

例外：

java.net.UnknownHostException

メソッド

close()

```
public void close()
```

送信スレッドをシャットダウンし、ソケットをクローズします。

定義：

インターフェイス AlarmWriter の close

getDescription()

```
public java.lang.String getDescription()
```

定義：

インターフェイス AlarmWriter の getDescription

戻り値：

AlarmWriter の簡単な説明。

getEnabled()

```
public boolean getEnabled()
```

定義：

インターフェイス AlarmWriter の getEnabled

戻り値：

AlarmWriter の有効状態。

getName()

```
public java.lang.String getName()
```

定義：

インターフェイス AlarmWriter の getName

戻り値：

AlarmWriter の名前。

main(String[])

```
public static void main(java.lang.String[] args)
```

send(String)

```
public void send(java.lang.String alarmMessage)
```

Alarm をアラーム サービスに送信します。

定義：

インターフェイス AlarmWriter の send

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

アプリケーションは、AlarmWriter を動的に有効または無効にできます。

定義：

インターフェイス AlarmWriter の setEnabled

ParameterList

ParameterList は、追加の（かつオプションの）ユーザ定義パラメータを AlarmService に送るのに使用される、名前と値のペアのリストです。これらのパラメータには、Alarm の詳細を保持できます。

たとえば、LowResourceAlarm には、次のように、どのリソースが低レベルであるかをサービスに通知するパラメータを保持できます。

name="CPUUsage"
value="0.9"

これらのパラメータはユーザ定義可能ですが、事前に AlarmService カタログに定義しておく必要があります。

宣言

```
public class ParameterList

java.lang.Object
      |
      +--com.cisco.services.alarm.ParameterList
```

メンバの概要

メンバの概要	
コンストラクタ	
	ParameterList() ParameterList のデフォルト コンストラクタ。
	ParameterList(String, String) 名前と値のペアを持つコンストラクタ。
メソッド	
void	addParameter(String, String) 名前と値のペア（パラメータ）をリストに追加するメソッド。
java.lang.String[]	getParameterNames() パラメータの値を取得します。
java.lang.String	getParameterValue(String) パラメータの値を取得します。
void	removeAllParameters() リストにあるすべてのパラメータを除去します。
void	removeParameter(String) 特定のパラメータを除去します（リストにある場合）。
java.lang.String	toString()

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

ParameterList()

```
public ParameterList()
```

ParameterList のデフォルト コンストラクタ。

ParameterList(String, String)

```
public ParameterList(java.lang.String name,
    java.lang.String value)
```

名前と値のペアを持つコンストラクタ。

メソッド

addParameter(String, String)

```
public void addParameter(java.lang.String name,
    java.lang.String value)
```

名前と値のペア（パラメータ）をリストに追加するメソッド。

getParameterNames()

```
public java.lang.String[] getParameterNames()
```

リストのパラメータ名を取得します。

戻り値：

パラメータの配列。

getParameterValue(String)

```
public java.lang.String getParameterValue(java.lang.String
    parameterName)
```

パラメータの値を取得します。

戻り値：

パラメータの値。

removeAllParameters()

```
public void removeAllParameters()
```

リストにあるすべてのパラメータを除去します。

removeParameter(String)

```
public void removeParameter(java.lang.String parameterName)
```

特定のパラメータを除去します（リストにある場合）。

toString()

```
public java.lang.String toString()
```

オーバーライド：

クラス Object 内の toString

アラーム インターフェイスの階層

次のインターフェイス階層は、com.cisco.services.alarm パッケージに含まれています。

com.cisco.services.alarm.Alarm

com.cisco.services.alarm.AlarmWriter

Alarm

Alarm インターフェイスは、Alarm の定義に使用します。アラームには、アラーム サービスにより認識されるために、次に示す DTD を使用した XML 表現に遵守する必要があります。アプリケーションでは、このインターフェイスを実装するかまたは AlarmFactory を使用して、正しいフォーマットの Alarm を生成できます。Alarm は AlarmService に送られる仕様であり、AlarmService は Alarm に基づいて必要な動作を実行します。この仕様を使用して、AlarmService はカタログで利用可能な定義にアクセスします。このカタログは、Alarm について適切な動作を実行する Alarm 関数を必要とするユーザによって維持されます。Alarm に指定した重大度は、カタログ内にあるこの Alarm に関連付けられた重大度をオーバーライドできます。Alarm に重大度を指定しない場合、カタログ重大度が使用されます。

アラームの重大度は、Syslog から取得され、次のように定義されます。

0 = EMERGENCIES、システム使用不可
1 = ALERTS、ただちに処置が必要
2 = CRITICAL、重大な状態
3 = ERROR、エラー状態
4 = WARNING、警告状態
5 = NOTIFICATION、動作は通常であるが重大な状態
6 = INFORMATIONAL、情報メッセージだけ
7 = DEBUGGING、デバッグ用メッセージ

宣言

```
public interface Alarm
```

既知の実装クラスの一覧

DefaultAlarm

メンバの概要

メンバの概要	
フィールド	
static int	ALERTS アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。 Syslog 重大度 = 1
static int	CRITICAL 重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。 Syslog 重大度 = 2
static int	DEBUGGING エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。 Syslog 重大度 = 7

メンバの概要 （続き）	
static int	EMERGENCIES 緊急状態であり、システムのシャットダウンが必要。 Syslog 重大度 = 0
static int	ERROR 何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。 Syslog 重大度 = 3
static int	HIGHEST_LEVEL 最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。
static int	INFORMATIONAL エラー、警告、監査、またはデバッグに関係しない形式の情報。 Syslog 重大度 = 6
static int	LOWEST_LEVEL 最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。
static int	NO_SEVERITY 重大度のない Alarm の生成には、アプリケーションでこのレベルを設定できる。
static int	NOTIFICATION NOTIFICATION は正常であるが、重大な状態を示す。 Syslog 重大度 = 5
static java.lang.String	UNKNOWN_MNEMONIC Alarm の送信の際、ニーモニックが設定されていない場合に使用される文字列。
static int	WARNING なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。 Syslog 重大度 = 4
メソッド	
java.lang.String	getFacility()
int	getSeverity()
java.lang.String	getSubFacility()
void	send(String) 指定されたニーモニック付きで Alarm を送信します。
void	send(String, ParameterList) 指定されたニーモニックとパラメータ リスト付きで Alarm を送信します。
void	send(String, String, String) 指定されたニーモニックと 1 つのパラメータ付きでアラームを送信します。

フィールド

ALERTS

```
public static final int ALERTS
```

アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。
Syslog 重大度 = 1

CRITICAL

```
public static final int CRITICAL
```

重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。
Syslog 重大度 = 2

DEBUGGING

```
public static final int DEBUGGING
```

エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。

Syslog 重大度 = 7

EMERGENCIES

```
public static final int EMERGENCIES
```

緊急状態であり、システムのシャットダウンが必要。
Syslog 重大度 = 0

ERROR

```
public static final int ERROR
```

何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。
Syslog 重大度 = 3

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。

INFORMATIONAL

```
public static final int INFORMATIONAL
```

エラー、警告、監査、またはデバッグに関係しない形式の情報。
Syslog 重大度 = 6

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。

NO_SEVERITY

```
public static final int NO_SEVERITY
```

重大度のない Alarm の生成には、アプリケーションでこのレベルを設定できる。注意：これは、アプリケーションで、カタログにあるアラームに関連する重大度を AlarmService に使用させる場合だけを想定しています。

NOTIFICATION

```
public static final int NOTIFICATION
```

NOTIFICATION は正常であるが、重大な状態を示す。

Syslog 重大度 = 5

UNKNOWN_MNEMONIC

```
public static final java.lang.String UNKNOWN_MNEMONIC
```

Alarm の送信の際、ニーモニックが設定されていない場合に使用される文字列。

WARNING

```
public static final int WARNING
```

なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。

Syslog 重大度 = 4

メソッド

getFacility()

```
public java.lang.String getFacility()
```

戻り値：

この Alarm のファシリティ名。

getSeverity()

```
public int getSeverity()
```

戻り値：

アラームの重大度。[0-7] の範囲の整数。

getSubFacility()

```
public java.lang.String getSubFacility()
```

戻り値：

この Alarm のサブファシリティ。

send(String)

```
public void send(java.lang.String mnemonic)
```

指定されたニーモニック付きで Alarm を送信します。null または空文字列が渡された場合、ニーモニック UNK が送信されます。

send(String, ParameterList)

```
public void send(java.lang.String mnemonic,
    com.cisco.services.alarm.ParameterList parameterList)
```

指定されたニーモニックとパラメータ リスト付きで Alarm を送信します。

send(String, String, String)

```
public void send(java.lang.String mnemonic,
    java.lang.String parameterName, java.lang.String parameterValue)
```

指定されたニーモニックと 1 つのパラメータ付きでアラームを送信します。

AlarmWriter

AlarmWriter は、アラーム メッセージを受け取り、TCP リンク上にある受信側の AlarmService にそのメッセージを転送します。このインターフェイスは、com.cisco.service.alarm の実装で使用する他の AlarmWriter を実装するために使用できます。DefaultAlarmWriter は、この実装に提供されていて、AlarmManager から取得できます。

宣言

```
public interface AlarmWriter
```

既知の実装クラスの一覧

DefaultAlarmWriter

メンバの概要

メンバの概要	
メソッド	
void	close() AlarmWriter をクローズします。
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
void	send(String) アラーム メッセージを AlarmService に送信します。
void	setEnabled(boolean) AlarmWriter を有効または無効にします。

メソッド

close()

```
public void close()
```

AlarmWriter をクローズします。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

AlarmWriter の説明。

getEnabled()

```
public boolean getEnabled()
```

戻り値：

AlarmWriter の現在の有効または無効の状態。

getName()

```
public java.lang.String getName()
```

戻り値：

AlarmWriter の名前。

send(String)

```
public void send(java.lang.String alarmMessage)
```

アラーム メッセージを AlarmService に送信します。

パラメータ：

`the`：送信する Alarm。

setEnabled(boolean)

```
public void setEnabled(boolean enable)
```

AlarmWriter を有効または無効にします。

パラメータ：

`enable`：AlarmWriter を有効または無効にします。

サービス トレース クラスの階層

次のクラス階層は、com.cisco.services.tracing パッケージに含まれています。

java.lang.Object

- com.cisco.services.tracing.BaseTraceWriter（com.cisco.services.tracing.TraceWriter を実装）
  - com.cisco.services.tracing.ConsoleTraceWriter
  - com.cisco.services.tracing.LogFileTraceWriter
  - com.cisco.services.tracing.OutputStreamTraceWriter
  - com.cisco.services.tracing.SyslogTraceWriter
- com.cisco.services.tracing.TraceManagerFactory

BaseTraceWriter

この抽象クラスは、デフォルトで非出力の `TraceWriter` を `TraceWriterManager` に提供するのに有効です。トレースする機能をさまざまなストリームに提供するには、このクラスを拡張する必要があります。拡張側のクラスによって、`doPrintln ()` メソッドを実装する必要があります。

宣言

```
public abstract class BaseTraceWriter implements TraceWriter
```

```
java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
```

実装インターフェイスの一覧

TraceWriter

直系の既知のサブクラス

ConsoleTraceWriter, LogFileTraceWriter, OutputStreamTraceWriter, SyslogTraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
protected	BaseTraceWriter(int[], String, String) Trace.LOWEST_LEVEL および Trace.HIGHEST_LEVEL の範囲を超えた traceLevels 配列に渡された、トレース レベル付き BaseTraceWriter は、無視されます。
protected	BaseTraceWriter(int, String, String) maxTraceLevel までのすべてのレベルをトレースする BaseTraceWriter。トレース レベルは、[Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL] の範囲に維持されます。
protected	BaseTraceWriter(String, String)) 重大度レベルが Trace.LOWEST_LEVEL である最低レベルのメッセージだけをトレースする BaseTraceWriter。
メソッド	
void	close()
protected void	doClose()
protected void	doFlush()
protected abstract void	doPrintln(String, int) 特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。
void	flush()
java.lang.String	getDescription()
boolean	getEnabled()
java.lang.String	getName()
int[]	getTraceLevels()
void	println(String, int)
void	setTraceLevels(int[])
java.lang.String	toString()

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

BaseTraceWriter(int[], String, String)

```
protected BaseTraceWriter(int[] traceLevels,
    java.lang.String name, java.lang.String description)
```

Trace.LOWEST_LEVEL および Trace.HIGHEST_LEVEL の範囲を超えた traceLevels 配列に渡された、トレース レベル付き BaseTraceWriter は、無視されます。

パラメータ：

traceLevels：トレース レベルの配列。

関連項目：

Trace

BaseTraceWriter(int, String, String)

```
protected BaseTraceWriter(int maxTraceLevel,
    java.lang.String name, java.lang.String description)
```

maxTraceLevel までのすべてのレベルをトレースする BaseTraceWriter。トレース レベルは、[Trace.HIGHEST_LEVEL, Trace.LOWEST_LEVEL] の範囲に維持されます。

関連項目：

Trace

BaseTraceWriter(String, String)

```
protected BaseTraceWriter(java.lang.String name,
    java.lang.String description)
```

重大度レベルが Trace.LOWEST_LEVEL である最低レベルのメッセージだけをトレースする BaseTraceWriter。

関連項目：

Trace

メソッド

close()

```
public final void close()
```

次のインターフェイスからコピーされた記述：

com.cisco.services.tracing.TraceWriter

この TraceWriter によって関連付けられたすべてのリソースを解放します。

定義：

インターフェイス TraceWriter の close

doClose()

```
protected void doClose()
```

doFlush()

```
protected void doFlush()
```

doPrintln(String, int)

```
protected abstract void doPrintln(java.lang.String message,
    int messageNumber)
```

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

flush()

```
public final void flush()
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

定義：

インターフェイス TraceWriter の flush

getDescription()

```
public final java.lang.String getDescription()
```

定義：

インターフェイス TraceWriter の getDescription

getEnabled()

```
public boolean getEnabled()
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

println メソッドによって出力されるものがあるかどうかを返します。クローズされた TraceWriter のこのメソッドは、常に false を返します。

定義：

インターフェイス TraceWriter の getEnabled

getName()

```
public final java.lang.String getName()
```

定義：

インターフェイス TraceWriter の getName

getTraceLevels()

```
public final int[] getTraceLevels()
```

定義：

インターフェイス TraceWriter の getTraceLevels

println(String, int)

```
public final void println(java.lang.String message, int severity)
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

指定された文字列とそれに続く改行を出力します。具象 TraceWriter クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。各 TraceWriter は、実行する必要のある最高位レベルのトレースを認知しています。

定義：

インターフェイス TraceWriter の println

setTraceLevels(int[])

```
public final void setTraceLevels(int[] levels)
```

次のインターフェイスからコピーされた記述：com.cisco.services.tracing.TraceWriter

この TraceWriter によりトレースされるトレース レベルを設定します。

定義：

インターフェイス TraceWriter の setTraceLevels

toString()

```
public final java.lang.String toString()
```

オーバーライド：

クラス Object 内の toString

ConsoleTraceWriter

トレースするコンソール TraceWriter を System.out. に提供します。

関連項目：
Trace

宣言

```
public final class ConsoleTraceWriter extends BaseTraceWriter
```

```
java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.ConsoleTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	ConsoleTraceWriter() デフォルトのコンストラクタ。すべての重大度レベルをトレースします。
	ConsoleTraceWriter(int) トレースする最大レベルを設定するコンストラクタ。
	ConsoleTraceWriter(int[]) トレース レベルの配列を使用して ConsoleTraceWriter を構築します。トレース レベル配列に重大度のあるトレースだけが、トレースされます。
メソッド	
protected void	doFlush()
protected void	doPrintln(String, int)
static void	main(String[])

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

ConsoleTraceWriter()

```
public ConsoleTraceWriter()
```

デフォルトのコンストラクタ。すべての重大度レベルをトレースします。

ConsoleTraceWriter(int)

```
public ConsoleTraceWriter(int maxTraceLevel)
```

トレースする最大レベルを設定するコンストラクタ。

関連項目：

Trace

ConsoleTraceWriter(int[])

```
public ConsoleTraceWriter(int[] traceLevels)
```

トレース レベルの配列を使用して ConsoleTraceWriter を構築します。トレース レベル配列に重大度のあるトレースだけが、トレースされます。

パラメータ：

int : [] traceLevels

関連項目：

Trace

メソッド

doFlush()

```
protected final void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected final void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述：com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

main(String[])

```
public static void main(java.lang.String[] args)
```

LogFileTraceWriter

このクラスは、BaseTraceWriter クラスを拡張します。ログ ファイルのセットに書き込みを行い、各ログ ファイルに指定された容量が満たされると、そのログ ファイルを指定のディレクトリに保存し、順番に次のファイルに書き込む TraceWriter が実装されます。

各ログ ファイルは、CurrentFile、FileNameBase、および FileExtension の 3 つのプロパティによって制御されるパターンに基づいて命名されます。CurrentFile プロパティは、序数でどのログ ファイルに現在書き込みが行われているのかを、FileNameBase プロパティは、各ログ ファイル名のプレフィクスを、また FileExtension プロパティは、「txt」などのサフィクスを決定します。これらのプロパティをもとに、ログ ファイルには **FileNameBase** *LeadingZeroPadding* CurrentFile.FileExtension という名前が割り当てられます。CurrentFile プロパティは、1 から MaxFiles プロパティまでの値を取ります。CurrentFile プロパティは、文字列に変換されると、MaxFiles および CurrentFile プロパティの値に応じて、先行するゼロでパディングされます。インデックス ファイルが、最後に書き込まれたファイルのインデックスを記録しています。logFileWriter が（アプリケーションがリスタートされた場合などで）再作成されると、最後の書き込みインデックスに続いて新規ファイルが作成されます。

ログ ファイルの保管場所は、path、dirNameBase、および useSameDir により決定されます。path を指定しない場合、デフォルトとして、現在のパスが使用されます。dirNameBase を指定しない場合、パスにログ ファイルが書き込まれます。LogFileTraceWriter インスタンスが作成されるたびに、useSameDir が true であるか false であるかに従って、ログ ファイルが同じディレクトリまたは

新規のディレクトリに書き込まれます。毎回新規のディレクトリが作成される場合、ディレクトリ名は、dirNameBase および「_」の後に番号が付いた名前になります。番号は、パスにある同じ dirNameBame のディレクトリに関連付けられた最大数よりも 1 大きい数です。パスの指定には、「/」または「¥¥」は使用できますが、「¥」は使用できません。

LogFileTraceWriter は、現在のログ ファイルに書き込まれたバイト数を記録しています。この数が LogFileTraceWriter.ROLLOVER_THRESHOLD バイトに到達すると次のファイルにログが書き込まれます。次のファイルは CurrentFile が MaxFiles に等しくない場合は CurrentFile + 1、または CurrentFile が MaxFiles に等しい場合は 1 です。

（注）　このクラスのすべてのプロパティは、コンストラクタに指定されます。これを動的に変更する方法はありません。**Caveat**：LogFileTraceWriter の 2 つのインスタンスが、同じ path と dirNameBase で作成され、useSameDir が true である場合、この 2 つは同じファイルに書き込まれる可能性があります。

例

次のコードでは、「MyLog01.log」から「MyLog12.log」のログ ファイルを作成する、LogFileTraceWriter をインスタンス化しています。各ファイルが、およそ 100K バイトのサイズまで大きくなると、次のファイルが作成されます。

LogFileTraceWriter out = new LogFileTraceWriter ("MyLog", "log", 12, 100 * 1024); と指定すると、ログ ファイルの TraceWriter が作成され、Mylog01.log から Mylog12.log までの 12 ファイルに、順番に 100K バイト分のトレースが書き込まれます。デフォルトでは、トレースは HIGHEST_LEVEL に設定されます。

例

次のコードでは、パス「c:/LogFiles」のサブディレクトリ「Run」にログ ファイルを保存する LogFileTraceWriter を構築します。ファイルは、MyLogXX.log. と命名されます。順に保存するファイル数は 12 で、サイズは 100 KB です。アプリケーションの各インスタンスで同じディレクトリを使用します。

LogFileTraceWriter out = new LogFileTraceWriter ("c:/logFiles", "Run", "MyLog", "log", 12, 100*1024, true);

関連項目：

Trace

宣言

```
public final class LogFileTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.LogFileTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
フィールド	
static java.lang.String	DEFAULT_FILE_NAME_BASE
static java.lang.String	DEFAULT_FILE_NAME_EXTENSION
static char	DIR_BASE_NAME_NUM_SEPERATOR
static int	MIN_FILE_SIZE
static int	MIN_FILES
static int	ROLLOVER_THRESHOLD
コンストラクタ	
	LogFileTraceWriter(String, String, int, int) すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。
	LogFileTraceWriter(String, String, String, String, int, int, boolean) すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。
	LogFileTraceWriter(String, String, String, String, int, int, int, boolean) 任意の数のファイルを順に処理し、指定のディレクトリに保存する LogFileTraceWriter を構築します。
メソッド	
protected void	doClose() この OutputStream をクローズします。
protected void	doFlush()
protected void	doPrintln(String, int)
int	getCurrentFile() CurrentFile プロパティ。
java.lang.String	getFileExtension() FileExtension プロパティ。
java.lang.String	getFileNameBase() FileNameBase プロパティ。
java.lang.String	getHeader() 各ログ ファイルの先頭に書き込まれるヘッダー文字列を取得します。
int	getMaxFiles() MaxFiles プロパティ。
int	getMaxFileSize() MaxFileSize プロパティ。
void	setHeader(String) 各ファイルの先頭に書き込まれるヘッダー定数を設定します。トレースの書き込みは、ヘッダーが書き込まれた次の行から継続されます。setHeader がファイル出力の開始後に呼び出された場合、次に書き込まれるファイルから有効になります。

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

フィールド

DEFAULT_FILE_NAME_BASE

```
public static final java.lang.String DEFAULT_FILE_NAME_BASE
```

DEFAULT_FILE_NAME_EXTENSION

```
public static final java.lang.String DEFAULT_FILE_NAME_EXTENSION
```

DIR_BASE_NAME_NUM_SEPERATOR

```
public static final char DIR_BASE_NAME_NUM_SEPERATOR
```

MIN_FILE_SIZE

```
public static final int MIN_FILE_SIZE
```

MIN_FILES

```
public static final int MIN_FILES
```

ROLLOVER_THRESHOLD

```
public static final int ROLLOVER_THRESHOLD
```

コンストラクタ

LogFileTraceWriter(String, String, int, int)

```
public LogFileTraceWriter(java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize)
    throws IOException
```

すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。path および Directory Base 名を指定しないので、ファイルは、サブディレクトリを作成せずに現行ディレクトリに書き込まれます。

例外：

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
    java.lang.String dirNameBase, java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize,
    boolean useSameDir)
    throws IOException
```

すべてのレベルのトレースに、任意の数のファイルを順に処理する LogFileTraceWriter のデフォルトのコンストラクタ。

例外：

java.io.IOException

LogFileTraceWriter(String, String, String, String, int, int, int, boolean)

```
public LogFileTraceWriter(java.lang.String path,
    java.lang.String dirNameBase, java.lang.String fileNameBase,
    java.lang.String fileNameExtension, int maxFiles, int maxFileSize,
    int maxTraceLevel, boolean useSameDir)
    throws IOException
```

任意の数のファイルを順に処理し、指定のディレクトリに保存する LogFileTraceWriter を構築します。

例外：

java.io.IOException

メソッド

doClose()

```
protected void doClose()
```

この OutputStream をクローズします。現在オープンされているすべてのログ ファイルも、同様にクローズされます。

オーバーライド：

クラス BaseTraceWriter 内の doClose

doFlush()

```
protected void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述：com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

getCurrentFile()

```
public int getCurrentFile()
```

戻り値：

CurrentFile プロパティ。

getFileExtension()

```
public java.lang.String getFileExtension()
```

戻り値：

FileExtension プロパティ。

getFileNameBase()

```
public java.lang.String getFileNameBase()
```

戻り値：

FileNameBase プロパティ。

getHeader()

```
public java.lang.String getHeader()
```

各ログ ファイルの先頭に書き込まれるヘッダー文字列を取得します。

戻り値：

Header プロパティ。

getMaxFiles()

```
public int getMaxFiles()
```

戻り値：

MaxFiles プロパティ。

getMaxFileSize()

```
public int getMaxFileSize()
```

戻り値：

MaxFileSize プロパティ。

setHeader(String)

```
public void setHeader(java.lang.String header)
```

各ファイルの先頭に書き込まれるヘッダー定数を設定します。トレースの書き込みは、ヘッダーが書き込まれた次の行から継続されます。setHeader がファイル出力の開始後に呼び出された場合、次に書き込まれるファイルから有効になります。

使用法：

```
tm = TraceManagerFactory.registerModule( this );
tw = new LogFileTraceWriter ( “trace”, “log”, 10, 1024*1024);
tw.setHeader ( header );
tm.getTraceWriterManager ().addTraceWriter (tw);
```

OutputStreamTraceWriter

OutputStreamTraceWriter は、TraceWriter の出力ストリームのラッパー クラスです。これにより、他の TraceWriter と共存可能なカスタム トレース クラスを簡単に追加できます。

宣言

```
public final class OutputStreamTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.OutputStreamTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	OutputStreamTraceWriter(int, OutputStream) オートフラッシュするデフォルトのコンストラクタ。
	OutputStreamTraceWriter(int, OutputStream, boolean) OutputStreamTraceWriter を作成します。
メソッド	
protected void	doClose()
protected void	doFlush()
protected void	doPrintln(String, int)
java.io.OutputStream	getOutputStream()

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

OutputStreamTraceWriter(int, OutputStream)

```
public OutputStreamTraceWriter(int maxTraceLevel,
    java.io.OutputStream outputStream)
```

オートフラッシュするデフォルトのコンストラクタ。

関連項目：

Trace

OutputStreamTraceWriter(int, OutputStream, boolean)

```
public OutputStreamTraceWriter(int maxTraceLevel,
    java.io.OutputStream outputStream, boolean autoFlush)
```

OutputStreamTraceWriter を作成します。

関連項目：

Trace

メソッド

doClose()

```
protected void doClose()
```

オーバーライド：

クラス BaseTraceWriter 内の doClose

doFlush()

```
protected void doFlush()
```

オーバーライド：

クラス BaseTraceWriter 内の doFlush

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
    int messageNumber)
```

次のクラスからコピーされた記述：com.cisco.services.tracing.BaseTraceWriter

特定のトレース機能を実現するために、BaseTraceWriter を拡張するさまざまな TraceWriter に実装する必要があります。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

getOutputStream()

```
public java.io.OutputStream getOutputStream()
```

戻り値：

TraceWriter に関連付けられた出力ストリーム。

SyslogTraceWriter

SyslogTraceWriter は、BaseTraceWriter を改良したもので、トレースを syslog に送ることが可能です。Cisco の syslog 仕様では、低レベルのトレースを UDP メッセージの形式で syslog コレクタに送ります。TraceWriter では、バッファリングは行われません。SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージ パケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。

宣言

```
public final class SyslogTraceWriter extends BaseTraceWriter

java.lang.Object
      |
      +--com.cisco.services.tracing.BaseTraceWriter
                |
                +--com.cisco.services.tracing.SyslogTraceWriter
```

実装インターフェイスの一覧

TraceWriter

メンバの概要

メンバの概要	
コンストラクタ	
	SyslogTraceWriter(int, String) 最大トレース レベル INFORMATIONAL の付けられたデフォルトの SyslogTraceWriter。
	SyslogTraceWriter(int, String, int) 最大トレース レベルの指定された SyslogTraceWriter。
	SyslogTraceWriter(int, String, int[]) トレース レベルの配列を指定できる SyslogTraceWriter。
メソッド	
void	doClose() ソケットをクローズします。
protected void	doPrintln(String, int) SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージ パケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。メッセージの、「¥r」または「¥n」より後の部分は無視されます。
static void	main(String[])

継承メンバの概要
クラス BaseTraceWriter から継承したメソッド
close(), doClose(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[]), toString()
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

コンストラクタ

SyslogTraceWriter(int, String)

```
public SyslogTraceWriter(int port, java.lang.String collector)
```

最大トレース レベル INFORMATIONAL の付けられたデフォルトの SyslogTraceWriter。

関連項目：

Trace

SyslogTraceWriter(int, String, int)

```
public SyslogTraceWriter(int port, java.lang.String collector,
    int maxTraceLevel)
```

最大トレース レベルの指定された SyslogTraceWriter。

関連項目：

Trace

SyslogTraceWriter(int, String, int[])

```
public SyslogTraceWriter(int port, java.lang.String collector,
    int[] traceLevels)
```

トレース レベルの配列を指定できる SyslogTraceWriter。

関連項目：

Trace

メソッド

doClose()

```
public void doClose()
```

ソケットをクローズします。

オーバーライド：

クラス BaseTraceWriter 内の doClose

doPrintln(String, int)

```
protected void doPrintln(java.lang.String message,
```

```
    int messageNumber)
```

SyslogTraceWriter には、println() メソッドに対して、システムで規定されているメッセージパケット末尾の行区切りの代わりに、「¥0」を使用する点で例外があります。メッセージの、「¥r」または「¥n」より後の部分は無視されます。

オーバーライド：

クラス BaseTraceWriter 内の doPrintln

main(String[])

```
public static void main(java.lang.String[] args)
```

TraceManagerFactory

TraceManagerFactory クラスは、アプリケーションが TraceManager オブジェクトの取得に使用するクラスです。コンストラクタに渡された TraceModule は、リストに登録されます。リストは、getModules() メソッドを使用して、列挙できます。

宣言

```
public class TraceManagerFactory

java.lang.Object
      |
      +--com.cisco.services.tracing.TraceManagerFactory
```

メンバの概要

メンバの概要	
メソッド	
static java.util.Enumeration	getModules() このファクトリに登録されている TraceModules の列挙を返します。
static TraceManager	registerModule(TraceModule) TraceManager オブジェクトのインスタンスを返します。
static TraceManager	registerModule(TraceModule, String[], TraceWriterManager)) TraceManager オブジェクトのインスタンスを返します。
static TraceManager	registerModule(TraceModule, TraceWriterManager) TraceManager オブジェクトのインスタンスを返します。

継承メンバの概要
クラス Object から継承したメソッド
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

メソッド

getModules()

```
public static java.util.Enumeration getModules()
```

このファクトリに登録されている TraceModules の列挙を返します。

registerModule(TraceModule)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module)
```

TraceManager オブジェクトのインスタンスを返します。包含される TraceWriterManager には、デフォルトの TraceWriter がありません。

registerModule(TraceModule, String[], TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module,
    java.lang.String[] subFacilities,
    com.cisco.services.tracing.TraceWriterManager traceWriterManager)
```

TraceManager オブジェクトのインスタンスを返します。Trace 出力は、指定された TraceWriterManager オブジェクトへリダイレクトされます。

registerModule(TraceModule, TraceWriterManager)

```
public static com.cisco.services.tracing.TraceManager
    registerModule(com.cisco.services.tracing.TraceModule module,
    com.cisco.services.tracing.TraceWriterManager traceWriterManager)
```

TraceManager オブジェクトのインスタンスを返します。Trace 出力は、指定された TraceWriterManager オブジェクトへリダイレクトされます。

サービス トレース インターフェイスの階層

次のインターフェイス階層は、com.cisco.services.tracing パッケージに含まれています。

com.cisco.services.tracing.Trace

　com.cisco.services.tracing.ConditionalTrace

　com.cisco.services.tracing.UnconditionalTrace

com.cisco.services.tracing.TraceManager

com.cisco.services.tracing.TraceModule

com.cisco.services.tracing.TraceWriter

com.cisco.services.tracing.TraceWriterManager

Trace

Trace インターフェイスは、アプリケーションのトレースを可能にするメソッドを定義します。Trace では、Syslog Trace Logging で指定される標準トレース タイプも定義しています。Syslog では、現在 8 レベルのトレースを定義しています。メッセージの重大度は、[0-7]（0 および 7 を含む）間

の範囲の数値で、トレース内に示されます。現在、7 が HIGHEST_LEVEL で、0 が LOWEST_LEVEL のトレースです。ここでは、トレース サブシステム上で参照される全 8 レベルを static int 型として事前定義しています。

次に、トレースされる重大度を示します。

0 = EMERGENCIES、システム使用不可
1 = ALERTS、ただちに処置が必要
2 = CRITICAL、重大な状態
3 = ERROR、エラー状態
4 = WARNING、警告状態
5 = NOTIFICATION、動作は通常であるが重大な状態
6 = INFORMATIONAL、情報メッセージだけ
7 = DEBUGGING、デバッグ用メッセージ

宣言

```
public interface Trace
```

すべての既知のサブインターフェイス

ConditionalTrace, UnconditionalTrace

メンバの概要

メンバの概要	
フィールド	
static int	ALERTS アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。 Syslog 重大度 = 1
static java.lang.String	ALERTS_TRACE_NAME ALERTS トレース レベルの文字列記述子。
static int	CRITICAL 重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。 Syslog 重大度 = 2
static java.lang.String	CRITICAL_TRACE_NAME CRITICAL トレース レベルの文字列記述子。
static int	DEBUGGING エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。 Syslog 重大度 = 7
static java.lang.String	DEBUGGING_TRACE_NAME DEBUGGING トレース レベルの文字列記述子。
static int	EMERGENCIES 緊急状態であり、システムのシャットダウンが必要。 Syslog 重大度 = 0

メンバの概要 （続き）	
static java.lang.String	EMERGENCIES_TRACE_NAME EMERGENCIES トレース レベルの文字列記述子。
static int	ERROR 何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。 Syslog 重大度 = 3
static java.lang.String	ERROR_TRACE_NAME ERROR トレース レベルの文字列記述子。
static int	HIGHEST_LEVEL 最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。
static int	INFORMATIONAL エラー、警告、監査、またはデバッグに関係しない形式の情報。 Syslog 重大度 = 6
static java.lang.String	INFORMATIONAL_TRACE_NAME INFORMATIONAL トレース レベルの文字列記述子。
static int	LOWEST_LEVEL 最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。
static int	NOTIFICATION NOTIFICATION は正常であるが、重大な状態を示す。 Syslog 重大度 = 5
static java.lang.String	NOTIFICATION_TRACE_NAME NOTIFICATION トレース レベルの文字列記述子。
static int	WARNING なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。 Syslog 重大度 = 4
static java.lang.String	WARNING_TRACE_NAME WARNING トレース レベルの文字列記述子。
メソッド	
java.lang.String	getName() この Trace オブジェクトの名前を返します。
java.lang.String	getSubFacility() トレースの subFacility のタイプ。
int	getType() Syslog に指定されているトレースのタイプ。
boolean	isEnabled() この Trace オブジェクトの状態を返します。
void	println(Object) Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。
void	println(String) Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。
void	println(String, Object) Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

メンバの概要 （続き）	
void	println(String, String) Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。
void	setDefaultMnemonic(String) このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

フィールド

ALERTS

```
public static final int ALERTS
```

アプリケーションはタスクの作業を継続できるが、すべての機能が動作可能なわけではない（リストの 1 つ以上のデバイスがアクセス不能であるが、他はアクセス可能）。
Syslog 重大度 = 1

ALERTS_TRACE_NAME

```
public static final java.lang.String ALERTS_TRACE_NAME
```

ALERTS トレース レベルの文字列記述子。

CRITICAL

```
public static final int CRITICAL
```

重大な障害であり、アプリケーションはこの障害が原因で要求されたタスクを達成できない。たとえば、アプリケーションが、データベースを開いてデバイス リストを読むことができない。
Syslog 重大度 = 2

CRITICAL_TRACE_NAME

```
public static final java.lang.String CRITICAL_TRACE_NAME
```

CRITICAL トレース レベルの文字列記述子。

DEBUGGING

```
public static final int DEBUGGING
```

エラーまたはプロセスの状態に関する詳細な情報で、DEBUG モードが有効になっているときだけに生成される。
Syslog 重大度 = 7

DEBUGGING_TRACE_NAME

```
public static final java.lang.String DEBUGGING_TRACE_NAME
```

DEBUGGING トレース レベルの文字列記述子。

EMERGENCIES

```
public static final int EMERGENCIES
```

緊急状態であり、システムのシャットダウンが必要。
Syslog 重大度 = 0

EMERGENCIES_TRACE_NAME

```
public static final java.lang.String EMERGENCIES_TRACE_NAME
```

EMERGENCIES トレース レベルの文字列記述子。

ERROR

```
public static final int ERROR
```

何らかのエラーの状態が発生し、ユーザは、この障害の性質を理解する必要がある。
Syslog 重大度 = 3

ERROR_TRACE_NAME

```
public static final java.lang.String ERROR_TRACE_NAME
```

ERROR トレース レベルの文字列記述子。

HIGHEST_LEVEL

```
public static final int HIGHEST_LEVEL
```

最高位のトレース レベルであり、現在は、トレース レベル 7 の DEBUGGING。

INFORMATIONAL

```
public static final int INFORMATIONAL
```

エラー、警告、監査、またはデバッグに関係しない形式の情報。
Syslog 重大度 = 6

INFORMATIONAL_TRACE_NAME

```
public static final java.lang.String INFORMATIONAL_TRACE_NAME
```

INFORMATIONAL トレース レベルの文字列記述子。

LOWEST_LEVEL

```
public static final int LOWEST_LEVEL
```

最低位のトレース レベルであり、現在は、トレース レベル 0 の EMERGENCIES。

NOTIFICATION

```
public static final int NOTIFICATION
```

NOTIFICATION は正常であるが、重大な状態を示す。
Syslog 重大度 = 5

NOTIFICATION_TRACE_NAME

```
public static final java.lang.String NOTIFICATION_TRACE_NAME
```

NOTIFICATION トレース レベルの文字列記述子。

WARNING

```
public static final int WARNING
```

なんらかの問題が存在するが、アプリケーションのタスクの実行が妨げられてはいないという警告。
Syslog 重大度 = 4

WARNING_TRACE_NAME

```
public static final java.lang.String WARNING_TRACE_NAME
```

WARNING トレース レベルの文字列記述子。

メソッド

getName()

```
public java.lang.String getName()
```

戻り値：

この Trace オブジェクトの名前を返します。

getSubFacility()

```
public java.lang.String getSubFacility()
```

戻り値：

トレースの subFacility のタイプ。

getType()

```
public int getType()
```

戻り値：

Syslog に指定されているトレースのタイプ。DEBUGGING、INFORMATIONAL、WARNING など。

isEnabled()

```
public boolean isEnabled()
```

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

println(Object)

```
public void println(java.lang.Object object)
```

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

パラメータ：

object：出力されるオブジェクト。

println(String)

```
public void println(java.lang.String message)
```

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

パラメータ：

message：出力されるメッセージ。

println(String, Object)

```
public void println(java.lang.String mnemonic,
    java.lang.Object object)
```

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

パラメータ：

object：出力されるオブジェクト。

mnemonic：出力されるメッセージにマップされるニーモニック。

println(String, String)

```
public void println(java.lang.String mnemonic,
    java.lang.String message)
```

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

パラメータ：

message：出力されるメッセージ。

mnemonic：出力されるメッセージにマップされるニーモニック。

setDefaultMnemonic(String)

```
public void setDefaultMnemonic(java.lang.String mnemonic)
```

このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

パラメータ：

mnemonic：ニーモニック文字列。

ConditionalTrace

ConditionalTrace インターフェイスは、Trace インターフェイスを拡張したもので、この特定条件のトレースの有効または無効を可能にするメソッドを定義しています。

一般に、アプリケーションは、条件当たり 1 つの ConditionalTrace オブジェクトを取得します。この条件は、常にではなく、特定の状況（たとえば、AUDIT、INFO など）においてトレースする必要があります。

宣言

```
public interface ConditionalTrace extends Trace
```

すべてのスーパーインターフェイス

Trace

メンバの概要

メンバの概要	
メソッド	
void	disable() このトレース条件を無効にします。
void	enable() このトレース条件を有効にします。

継承メンバの概要
インターフェイス Trace から継承したフィールド
ALERTS, ALERTS_TRACE_NAME, CRITICAL, CRITICAL_TRACE_NAME, DEBUGGING, DEBUGGING_TRACE_NAME, EMERGENCIES, EMERGENCIES_TRACE_NAME, ERROR, ERROR_TRACE_NAME, HIGHEST_LEVEL, INFORMATIONAL, INFORMATIONAL_TRACE_NAME, LOWEST_LEVEL, NOTIFICATION, NOTIFICATION_TRACE_NAME, WARNING, WARNING_TRACE_NAME
インターフェイス Trace から継承したメソッド
getName(), getSubFacility(), getType(), isEnabled(), println(Object), println(String), println(String, Object), println(String, String), setDefaultMnemonic(String)

メソッド

disable()

```
public void disable()
```

このトレース条件を無効にします。

enable()

```
public void enable()
```

このトレース条件を有効にします。

UnconditionalTrace

UnconditionalTrace インターフェイスは、Trace インターフェイスを拡張したものです。このオブジェクトは Trace を拡張しているため、その状態はデフォルトで有効になっていて、変更できないことに注意してください。

一般に、アプリケーションは、条件当たり 1 つの UnconditionalTrace オブジェクトを取得します。この条件は、常にすべての状況（ERROR、FATAL など）においてトレースする必要があります。

宣言

```
public interface UnconditionalTrace extends Trace
```

すべてのスーパーインターフェイス

Trace

メンバの概要

継承メンバの概要
インターフェイス Trace から継承したフィールド
ALERTS, ALERTS_TRACE_NAME, CRITICAL, CRITICAL_TRACE_NAME, DEBUGGING, DEBUGGING_TRACE_NAME, EMERGENCIES, EMERGENCIES_TRACE_NAME, ERROR, ERROR_TRACE_NAME, HIGHEST_LEVEL, INFORMATIONAL, INFORMATIONAL_TRACE_NAME, LOWEST_LEVEL, NOTIFICATION, NOTIFICATION_TRACE_NAME, WARNING, WARNING_TRACE_NAME
インターフェイス Trace から継承したメソッド
getName(), getSubFacility(), getType(), isEnabled(), println(Object), println(String), println(String, Object), println(String, String), setDefaultMnemonic(String)

TraceManager

TraceManager インターフェイスは、アプリケーションのトレース管理を可能にするメソッドを定義します。

一般に、アプリケーションは、1 つの TraceManager オブジェクトだけを取得します。すべての Trace オブジェクトがデフォルトで作成されます。次に、Syslog の定義に基づいた事前定義の Trace を示します。

```
ConditionalTraces: INFORMATIONAL, DEBUGGING, NOTIFICATION, WARNING
UnconditionalTraces: ERROR, CRITICAL, ALERTS, EMERGENCIES
```

ファシリティ /サブファシリティ：

- **Facility**：メッセージが参照するファシリティを示す、2 つ以上の大文字からなるコードです。ファシリティは、ハードウェア デバイス、プロトコル、またはシステム ソフトウェアのモジュールなどです。

- **SubFacility**：メッセージが参照するサブファシリティを示す、2 つ以上の大文字からなるコードです。サブファシリティは、ハードウェア デバイス コンポーネント、プロトコル ユニット、またはシステム ソフトウェアのサブモジュールなどです。

デフォルトでは、8 つの Conditional と UnConditional Trace がすべて Facility に作成されます。各 subFacility にも 8 つ作成されます。たとえば、親 FACILITY で DEBUGGING トレースを使用するには、アプリケーションは、このオブジェクトの getConditionalTrace("DEBUGGING") メソッドを使用する必要があります。

たとえば、SUBFACILITY で DEBUGGING トレースを使用するには、アプリケーションで、このオブジェクトの getConditionalTrace(SUBFACILITY + "_" + "DEBUGGING") メソッドを使用するか、getConditionalTrace(SUBFACILITY, "DEBUGGING") メソッドを使用する必要があります。

システム全体の TraceWriterManager は、このインターフェイスが提供する setTraceWriterManager メソッドにより設定されます。

また、Trace Manager オブジェクトでは、enableAll() および disableAll() メソッドを使用して、すべてのトレースをアプリケーションによって有効または無効にすることが可能です。

宣言

```
public interface TraceManager
```

メンバの概要

メンバの概要	
メソッド	
void	addSubFacilities(String[]) この TraceManager/Facility の sub-Facility のセットを設定します。
void	addSubFacility(String) この TraceManager/Facility の単一の subFacility を追加します。
void	disableAll() この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。
void	disableTimeStamp() この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。
void	enableAll() この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。
void	enableTimeStamp() この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。
ConditionalTrace	getConditionalTrace(int) この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。
ConditionalTrace	getConditionalTrace(String, int) この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

メンバの概要 （続き）	
java.lang.String	getName() この TraceManager の Facility 名を返します。
java.lang.String[]	getSubFacilities() この TraceManager/Facility の subFacility 名を返します。
java.util.Enumeration	getTraces() この TraceManager により管理される Trace オブジェクトの列挙を返します。
TraceWriterManager	getTraceWriterManager() この TraceManager により使用される TraceWriter を返します。
UnconditionalTrace	getUnconditionalTrace(int) この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。
UnconditionalTrace	getUnconditionalTrace(String, int) この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。
void	removeTrace(Trace) Trace オブジェクトを削除します（オブジェクトがある場合）。
void	setSubFacilities(String[]) この TraceManager/Facility の sub-Facility のセットを設定します。
void	setSubFacility(String) この TraceManager/Facility の単一の subFacility を追加します。
void	setTraceWriterManager(TraceWriterManager) この TraceManager で使用する TraceWriter を設定します。

メソッド

addSubFacilities(String[])

```
public void addSubFacilities(java.lang.String[] names)
```

この TraceManager/Facility の sub-Facility のセットを設定します。

addSubFacility(String)

```
public void addSubFacility(java.lang.String name)
```

この TraceManager/Facility の単一の subFacility を追加します。

disableAll()

```
public void disableAll()
```

この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。

disableTimeStamp()

```
public void disableTimeStamp()
```

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。

enableAll()

```
public void enableAll()
```

この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。

enableTimeStamp()

```
public void enableTimeStamp()
```

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。

getConditionalTrace(int)

```
public com.cisco.services.tracing.ConditionalTrace
    getConditionalTrace(int severity)
```

この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

getConditionalTrace(String, int)

```
public com.cisco.services.tracing.ConditionalTrace
    getConditionalTrace(java.lang.String subFacility, int severity)
```

この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

getName()

```
public java.lang.String getName()
```

この TraceManager の Facility 名を返します。

getSubFacilities()

```
public java.lang.String[] getSubFacilities()
```

この TraceManager/Facility の subFacility 名を返します。

getTraces()

```
public java.util.Enumeration getTraces()
```

この TraceManager により管理される Trace オブジェクトの列挙を返します。

getTraceWriterManager()

```
public com.cisco.services.tracing.TraceWriterManager
    getTraceWriterManager()
```

この TraceManager により使用される TraceWriter を返します。

getUnconditionalTrace(int)

```
public com.cisco.services.tracing.UnconditionalTrace
    getUnconditionalTrace(int severity)
```

この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

getUnconditionalTrace(String, int)

```
public com.cisco.services.tracing.UnconditionalTrace
    getUnconditionalTrace(java.lang.String subFacility, int severity)
```

この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

removeTrace(Trace)

```
public void removeTrace(com.cisco.services.tracing.Trace tc)
```

Trace オブジェクトを削除します（オブジェクトがある場合）。

setSubFacilities(String[])

```
public void setSubFacilities(java.lang.String[] names)
```

推奨されません。

TraceManager.addSubFacilities メソッドに置き換えられました。

この TraceManager/Facility の sub-Facility のセットを設定します。

setSubFacility(String)

```
public void setSubFacility(java.lang.String name)
```

推奨されません。

TraceManager.addSubFacility メソッドに置き換えられました。

この TraceManager/Facility の単一の subFacility を追加します。

setTraceWriterManager(TraceWriterManager)

```
public void
    setTraceWriterManager(com.cisco.services.tracing.TraceWriterManager twm)
```

この TraceManager で使用する TraceWriter を設定します。

TraceModule

TraceModule インターフェイスは、2 つの目的で使用されます。第 1 に、アプリケーション上で使用中の他のパッケージが使用している TraceManager オブジェクトを、アプリケーションが検出できるようにします。第 2 に、TraceManagerFactory に登録するアプリケーションは、このインターフェイスを実装して、アプリケーション自体の ID を示す必要があります。

宣言

```
public interface TraceModule
```

すべての既知のサブインターフェイス

com.cisco.jtapi.extensions.CiscoJtapiPeer

メンバの概要

メンバの概要	
メソッド	
TraceManager	getTraceManager() オブジェクトがトレースに使用している TraceManager を返します。
java.lang.String	getTraceModuleName() モジュール名を返します。

メソッド

getTraceManager()

```
public com.cisco.services.tracing.TraceManager getTraceManager()
```

オブジェクトがトレースに使用している TraceManager を返します。

getTraceModuleName()

```
public java.lang.String getTraceModuleName()
```

モジュール名を返します。

TraceWriter

TraceWriter インターフェイスは、トレース メッセージ出力の abstruct クラスです。TraceWriter は、enabled メソッドを使用して、print メソッドや println メソッドでの出力を有効化・無効化します。TraceWriter のユーザは、print メソッドおよび println メソッドを呼び出す必要があるかどうかの指標として、getEnabled メソッドが返す値を使用する必要があります。

宣言

```
public interface TraceWriter
```

すべての既知のサブインターフェイス

TraceWriterManager

既知の実装クラスの一覧

BaseTraceWriter

メンバの概要

メンバの概要	
メソッド	
void	close() この TraceWriter によって関連付けられたすべてのリソースを解放します。
void	flush() println メソッドを使用して、出力されたすべてのメッセージを強制出力します。
java.lang.String	getDescription()
boolean	getEnabled() println メソッドによって出力されるものがあるかどうかを返します。
java.lang.String	getName()
int[]	getTraceLevels()
void	println(String, int) 指定された文字列とそれに続く改行を出力します。TraceWriter の具象 クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。
void	setTraceLevels(int[]) この TraceWriter によりトレースされるトレース レベルを設定します。

メソッド

close()

```
public void close()
```

この TraceWriter によって関連付けられたすべてのリソースを解放します。

flush()

```
public void flush()
```

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

getDescription()

```
public java.lang.String getDescription()
```

戻り値：

この TraceWriter の簡単な説明。

getEnabled()

```
public boolean getEnabled()
```

println メソッドによって出力されるものがあるかどうかを返します。クローズされた TraceWriter のこのメソッドは、常に false を返します。

戻り値：

TraceWriter が有効の場合に true、そうでない場合に false。

getName()

```
public java.lang.String getName()
```

戻り値：

この TraceWriter の名前を返します。

getTraceLevels()

```
public int[] getTraceLevels()
```

戻り値：

この TraceWriter によりトレースされるトレース レベルの配列。

println(String, int)

```
public void println(java.lang.String message, int severity)
```

指定された文字列とそれに続く改行を出力します。TraceWriter の具象クラスでは、重大度を使用して、特定のストリームからのメッセージをブロックします。各 TraceWriter は、自分が実行する必要のある最高位レベルのトレース レベルを認知しています。

パラメータ：

`message`：出力されるメッセージ。

`severity`：トレースの重大度。

関連項目：

Trace

setTraceLevels(int[])

```
public void setTraceLevels(int[] levels)
```

この TraceWriter によりトレースされるトレース レベルを設定します。

パラメータ：

`int[]`：レベル。

関連項目：

Trace

TraceWriterManager

TraceWriterManager には、トレースの実装に使用される TraceWriter オブジェクトのリストを保持します。リストは、起動時に読み込まれる .ini ファイルの設定によって初期化されます。LogFileTraceWriter、ConsoleTraceWriter および SyslogTraceWriter が利用可能です。ユーザは、ユーザ実装の TraceWriter[] を設定して、あるいは既存の TraceWriter に追加して、既存の TraceWriter をオーバーライドすることができます。これにより、既存の TraceWriter と併存して機能する、他の TraceWriter を追加することが可能です。

宣言

```
public interface TraceWriterManager extends TraceWriter
```

すべてのスーパーインターフェイス

TraceWriter

メンバの概要

メンバの概要	
メソッド	
void	addTraceWriter(TraceWriter) 別の TraceWriter を配列に追加します。
TraceWriter[]	getTraceWriters()
void	removeTraceWriter(TraceWriter) Manager の配列から TraceWriter を取り除きます。
void	setTraceWriters(TraceWriter[]) 実装では、このメソッドを使用して、提供された TraceWriter をオーバーライドまたは拡張できます。

継承メンバの概要
インターフェイス TraceWriter から継承したメソッド
close(), flush(), getDescription(), getEnabled(), getName(), getTraceLevels(), println(String, int), setTraceLevels(int[])

メソッド

addTraceWriter(TraceWriter)

```
public void
    addTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

別の TraceWriter を配列に追加します。

パラメータ：

TraceWriter：リストに追加される TraceWriter。

getTraceWriters()

```
public com.cisco.services.tracing.TraceWriter[] getTraceWriters()
```

戻り値：

Manager の TraceWriter の配列。

removeTraceWriter(TraceWriter)

```
public void
    removeTraceWriter(com.cisco.services.tracing.TraceWriter traceWriter)
```

Manager の配列から TraceWriter を取り除きます。

setTraceWriters(TraceWriter[])

```
public void
    setTraceWriters(com.cisco.services.tracing.TraceWriter[] traceWriters)
```

実装では、このメソッドを使用して、提供された TraceWriter をオーバーライドまたは拡張できます。

パラメータ：

`set`：TraceWriter の配列を設定します。

トレースの実装クラスの階層

次のトレースの実装クラスの階層は、com.cisco.services.tracing.implementation パッケージに含まれています。

java.lang.Object

com.cisco.services.tracing.implementation.TraceImpl（com.cisco.services.tracing.Trace を実装）

com.cisco.services.tracing.implementation.ConditionalTraceImpl（com.cisco.services.tracing.ConditionalTrace を実装）

com.cisco.services.tracing.implementation.UnconditionalTraceImpl（com.cisco.services.tracing.UnconditionalTrace を実装）

com.cisco.services.tracing.implementation.TraceManagerImpl（com.cisco.services.tracing.TraceManager を実装）

com.cisco.services.tracing.implementation.TraceWriterManagerImpl（com.cisco.services.tracing.TraceWriterManager を実装）

TraceImpl

宣言

public abstract class **TraceImpl**

extends java.lang.Object

implements Trace

実装インターフェイスの一覧

Trace

メソッド

println

public final void **println**(java.lang.String message)

次のインターフェイスからコピーされた記述：Trace

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

message：出力されるメッセージ。

println

public final void println(java.lang.String mnemonic,
java.lang.String message)

次のインターフェイスからコピーされた記述：Trace

Trace.print() と同じフォーマットでメッセージを出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

mnemonic：出力されるメッセージにマップされるニーモニック。

message：出力されるメッセージ。

println

public final void **println**(java.lang.Object object)

次のインターフェイスからコピーされた記述：Trace

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

object：出力されるオブジェクト。

println

public final void **println**(java.lang.String mnemonic,
java.lang.Object object)

次のインターフェイスからコピーされた記述：Trace

Object.toString() メソッドによって返された文字列を出力し、システムでの定義に従って行を終了します。

定義：

インターフェイス Trace の println

パラメータ：

mnemonic：出力されるメッセージにマップされるニーモニック。

object：出力されるオブジェクト。

getName

public final java.lang.String **getName**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの名前を返します。

定義：

インターフェイス Trace の getName

戻り値：

この Trace オブジェクトの名前を返します。

setDefaultMnemonic

public final void **setDefaultMnemonic**(java.lang.String mnemonic)

次のインターフェイスからコピーされた記述：Trace

このトレースに出力されるすべてのメッセージにデフォルトのニーモニックを設定します。

定義：

インターフェイス Trace の setDefaultMnemonic

パラメータ：

mnemonic：ニーモニック文字列。

getType

public int **getType**()

次のインターフェイスからコピーされた記述：Trace

トレースのタイプを返します。

定義：

インターフェイス Trace の getType

戻り値：

Syslog に指定されているトレースの重大度。DEBUGGING、INFORMATIONAL、WARNING など。

getSubFacility

public java.lang.String **getSubFacility**()

次のインターフェイスからコピーされた記述：Trace

トレースの subFacility を返します。

定義：

インターフェイス Trace の getSubFacility

戻り値：

トレースの subFacility のタイプ。

継承したメソッド

isEnabled()

ConditionalTraceImpl

宣言

public final class **ConditionalTraceImpl**
extends TraceImpl
implements ConditionalTrace

実装インターフェイスの一覧

ConditionalTrace、Trace

メソッド

enable

public void **enable**()

次のインターフェイスからコピーされた記述：ConditionalTrace

このトレース条件を有効にします。

定義：

インターフェイス ConditionalTrace の enable

disable

public void **disable**()

次のインターフェイスからコピーされた記述：ConditionalTrace

このトレース条件を無効にします。

定義：

インターフェイス ConditionalTrace の disable

isEnabled()

public boolean **isEnabled**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

定義：

インターフェイス Trace の isEnabled

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

UnconditionalTraceImpl

宣言

public final class **UnconditionalTraceImpl**

extends TraceImpl

implements UnconditionalTrace

実装インターフェイスの一覧

Trace、UnconditionalTrace

メソッド

isEnabled()

public boolean **isEnabled**()

次のインターフェイスからコピーされた記述：Trace

この Trace オブジェクトの状態を返します。デフォルトでは、Trace オブジェクトは有効になっています。つまり、常に println() メソッドのトレースを実行します。このインターフェイスでは状態を変更できませんが、このオブジェクトに、状態を変更可能な追加のインターフェイスを実装することができます。

定義：

インターフェイス Trace の isEnabled

戻り値：

トレースが有効の場合に true、そうでない場合に false。

関連項目：

ConditionalTrace

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

TraceManagerImpl

TraceManagerImpl クラスは TraceManager インターフェイスを実装しています。

宣言

public class **TraceManagerImpl** extends java.lang.Object

java.lang.Object
|
+--com.cisco.services.tracing.TraceManagerImpl

実装インターフェイスの一覧

TraceManager

コンストラクタ

public **TraceManagerImpl**(java.lang.String moduleName, java.lang.String[] subFacilities, TraceWriterManager traceWriterManager)

public **TraceManagerImpl**(java.lang.String moduleName, TraceWriterManager traceWriterManager)

メソッド

getConditionalTrace

public ConditionalTrace **getConditionalTrace**(int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getConditionalTrace

getConditionalTrace

public ConditionalTrace getConditionalTrace(java.lang.String subFacility, int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件および subFacility について、新規の ConditionalTrace オブジェクトを作成する、または既存の ConditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getConditionalTrace

getUnconditionalTrace

public UnconditionalTrace getUnconditionalTrace(int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getUnconditionalTrace

getUnconditionalTrace

public UnconditionalTrace getUnconditionalTrace(java.lang.String subFacility, int severity)

次のインターフェイスからコピーされた記述：TraceManager

この条件および subFacility について、新規の UnconditionalTrace オブジェクトを作成するか、または既存の UnconditionalTrace オブジェクトを取得します。

定義：

インターフェイス TraceManager の getUnconditionalTrace

getTraceWriterManager

public TraceWriterManager getTraceWriterManager()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により使用される TraceWriter を返します。

定義：

インターフェイス TraceManager の getTraceWriterManager

setTraceWriterManager

public void setTraceWriterManager(TraceWriterManager out)

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager で使用する TraceWriter を設定します。

定義：

インターフェイス TraceManager の setTraceWriterManager

removeTrace

public void removeTrace(Trace tc)

次のインターフェイスからコピーされた記述：TraceManager

Trace オブジェクトを削除します（オブジェクトがある場合）。

定義：

インターフェイス TraceManager の removeTrace

getTraces()

public java.util.Enumeration getTraces()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により管理される Trace オブジェクトの列挙を返します。

定義：

インターフェイス TraceManager の getTraces

enableAll

public void enableAll()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって管理されるすべての Trace オブジェクトのトレースを有効にします。

定義：

インターフェイス TraceManager の enableAll

disableAll

public void disableAll()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager により管理されるすべての Trace オブジェクトのトレースを無効にします。

定義：

インターフェイス TraceManager の disableAll

getName

public java.lang.String getName()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager の Facility 名を返します。

定義：

インターフェイス TraceManager の getName

enableTimeStamp

public void enableTimeStamp()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを有効にします。

定義：

インターフェイス TraceManager の enableTimeStamp

disableTimeStamp

public void disableTimeStamp()

次のインターフェイスからコピーされた記述：TraceManager

この TraceManager によって出力されるすべてのメッセージについて、タイムスタンプにプレフィクスを付けることを無効にします。

定義：

インターフェイス TraceManager の disableTimeStamp

getSubFacilities

public java.lang.String[] getSubFacilities()

この TraceManager/Facility の subFacility 名を返します。

定義：

インターフェイス TraceManager の getSubFacilities

addSubFacilities

public void addSubFacilities(java.lang.String[] names)

この TraceManager/Facility の subFacility を追加します。

定義：

インターフェイス TraceManager の addSubFacilities

addSubFacility

public void addSubFacility(java.lang.String name)

この TraceManager/Facility の subFacility を追加します。

定義：

インターフェイス TraceManager の addSubFacility

非推奨のメソッド

getSubFacilities(java.lang.String[] names)

addSubFacilties(String[]) に置き換えられました。

setSubFacility(java.lang.String name)

addSubFacility(String) に置き換えられました。

継承したメソッド

クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

TraceWriterManagerImpl

TraceWriterManager には、トレースの実装に使用される TraceWriter オブジェクトのリストを保持します。リストは、始動時に、.ini ファイル内のスイッチによって入力されます。LogFileTraceWriter、ConsoleTraceWriter および SyslogTraceWriter が利用可能です。ユーザは、ユーザ実装の TraceWriter[] を設定して、あるいは既存の TraceWriter に追加して、既存の TraceWriter をオーバーライドすることができます。これにより、既存の TraceWriter と併存して機能する、他の TraceWriter を追加することが可能です。

（注） クラス java.lang.Object から継承したメソッドは、clone、equals、finalize、getClass、hashCode、notify、notifyAll、toString、wait、wait、wait です。

宣言

public class TraceWriterManagerImpl extends java.lang.Object implements TraceWriterManager

java.lang.Object

com.cisco.services.tracing.implementation.TraceWriterManagerImpl

実装インターフェイスの一覧

TraceWriter、TraceWriterManager

コンストラクタ

TraceWriterManagerImpl

public TraceWriterManagerImpl()

長さゼロの TraceWriter 配列で TraceWriterManagerImpl を作成します。

メソッド

setTraceWriters

public void setTraceWriters(TraceWriter[] traceWriters)

既存の TraceWriter を新しいユーザが指定したセットで上書きします。

定義：

インターフェイス TraceWriterManager の setTraceWriters

パラメータ：

traceWriters：TraceWriter の配列です。

getTraceWriters()

public TraceWriter[] getTraceWriters()

現在使用されている TraceWriter の配列を返します。

定義：

インターフェイス TraceWriterManager の getTraceWriters

戻り値：

Manager の TraceWriter の配列。

addTraceWriter

public void addTraceWriter(TraceWriter tw)

この TraceWriter を TraceWriter の配列に追加します。

定義：

インターフェイス TraceWriterManager の addTraceWriter

パラメータ：

tw：リストに追加される TraceWriter です。

removeTraceWriter

public void removeTraceWriter(TraceWriter tw)

TraceWriter の配列から Tracewriter を削除します。

定義：

インターフェイス TraceWriterManager の removeTraceWriter

println

public void println(java.lang.String message, int severity)

すべてのトレースがこのメソッドを呼び出します。トレースによって、メッセージとともに重大度が指定されます。トレースが TraceWriter の重大度のしきい値以下になる場合があります。例：重大度のしきい値が INFORMATIONAL (level = 6) DEBUG に設定されている場合、トレースは TraceWriter によって渡されません。重大度レベルは TraceWriter のコンストラクタで設定されます。

定義：

インターフェイス TraceWriter の println

パラメータ：

message：出力されるメッセージ。

severity：トレースの重大度。

関連項目：

Trace

フラッシュ

public void flush()

次のインターフェイスからコピーされた記述：TraceWriter

println メソッドを使用して、出力されたすべてのメッセージを強制出力します。

定義：

インターフェイス TraceWriter の flush

close()

public void close()

次のインターフェイスからコピーされた記述：TraceWriter

この TraceWriter によって関連付けられたすべてのリソースを解放します。

定義：

インターフェイス TraceWriter の close

getEnabled

public boolean getEnabled()

基になる TraceWriter のいずれかが有効になっている場合は true を返し、そうでない場合は false を返します。

定義：

インターフェイス TraceWriter の getEnabled

戻り値：

TraceWriter が有効の場合に true、そうでない場合に false。

getName

public java.lang.String getName()

定義：

インターフェイス TraceWriter の getName

戻り値：

この TraceWriter の名前を返します。

getDescription

public java.lang.String getDescription()

定義：

インターフェイス TraceWriter の getDescription

戻り値：

この TraceWriter の簡単な説明。

setTraceLevels

public void setTraceLevels(int[] levels)

TraceWriterManager はこのメソッドに対して何も行いません。

定義：

インターフェイス TraceWriter の setTraceLevels

パラメータ：

Levels：トレース レベルの配列。

関連項目：

Trace

getTraceLevels

public int[] getTraceLevels()

traceLevel が個々の TraceWriter で維持されるため、TraceWriterManager は null を返します。

定義：

インターフェイス TraceWriter の getTraceLevels

戻り値：

null

■ TraceWriterManagerImpl

CHAPTER 8

Cisco Unified JTAPI の例

この章では、makecall（JTAPI インストールをテストするのに使用される Cisco Unified JTAPI プログラム）のソース コードを記載しています。makecall プログラムは、Cisco Unified JTAPI 実装を使用して Java で書かれた一連のプログラムから構成されています。

この章は次のセクションで構成されています。

- MakeCall.java
- Actor.java
- Originator.java
- Receiver.java
- StopSignal.java
- Trace.java
- TraceWindow.java

この章では、makecall を呼び出す方法の説明も載せています。

- makecall の実行

MakeCall.java

```
/**
 * makecall.java
 *
 * Copyright Cisco Systems, Inc.
 *
 * Performance-testing application (first pass) for Cisco JTAPI
 * implementation.
 *
 * Known problems:
 *
 * Due to synchronization problems between Actors, calls may
 * not be cleared when this application shuts down.
 *
 */

//import com.ms.wfc.app.*;
import java.util.*;
import javax.telephony.*;
import javax.telephony.events.*;
import com.cisco.cti.util.Condition;

public class makecall extends TraceWindow implements ProviderObserver
```

MakeCall.java

```
{
    Vector  actors = new Vector ();
    ConditionconditionInService = new Condition ();
    Providerprovider;

    public makecall ( String [] args ) {

        super ( "makecall" + ": "+ new CiscoJtapiVersion());
        try {

            println ( "Initializing Jtapi" );
            int curArg = 0;
            String providerName = args[curArg++];
            String login = args[curArg++];
            String passwd = args[curArg++];
            int actionDelayMillis = Integer.parseInt ( args[curArg++] );
            String src = null;
            String dest = null;

            JtapiPeer peer = JtapiPeerFactory.getJtapiPeer ( null );
            if ( curArg < args.length ) {

                String providerString = providerName + ";login=" + login + ";passwd=" + passwd;
                println ( "Opening " + providerString + "...¥n" );
                provider = peer.getProvider ( providerString );
                provider.addObserver ( this );
                conditionInService.waitTrue ();

                println ( "Constructing actors" );

                for ( ; curArg < args.length; curArg++ ) {
                    if ( src == null ) {
                        src = args[curArg];
                    }
                    else {
                        dest = args[curArg];
                        Originator originator = new Originator ( provider.getAddress ( src ), dest, this,
actionDelayMillis );
                        actors.addElement ( originator );
                        actors.addElement (
                            new Receiver ( provider.getAddress ( dest ), this, actionDelayMillis, originator )
                            );
                        src = null;
                        dest = null;
                    }
                }
                if ( src != null ) {
                    println ( "Skipping last originating address ¥"" + src + "¥"; no destination specified" );
                }

            }

            Enumeration e = actors.elements ();
            while ( e.hasMoreElements () ) {
                Actor actor = (Actor) e.nextElement ();
                actor.initialize ();
            }

            Enumeration en = actors.elements ();
            while ( en.hasMoreElements () ) {
                Actor actor = (Actor) en.nextElement ();
                actor.start ();
            }
        }
```

```
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
    }

    public void dispose () {
        println ( "Stopping actors" );
        Enumeration e = actors.elements ();
        while ( e.hasMoreElements () ) {
            Actor actor = (Actor) e.nextElement ();
            actor.dispose ();
        }
    }

    public static void main ( String [] args )
    {
        if ( args.length < 6 ) {
            System.out.println ( "Usage: makecall <server> <login> <password> <delay> <origin> <destination>
..." );
            System.exit ( 1 );
        }
        new makecall ( args );
    }

    public void providerChangedEvent ( ProvEv [] eventList ) {
        if ( eventList != null ) {
            for ( int i = 0; i < eventList.length; i++ )
            {
                if ( eventList[i] instanceof ProvInServiceEv ) {
                    conditionInService.set ();
                }
            }
        }
    }
}
```

Actor.java

```
/**
 * Actor.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.cisco.jtapi.extensions.*;
public abstract class Actor implements AddressObserver, TerminalObserver, CallControlCallObserver, Trace
{

    public static final int ACTOR_OUT_OF_SERVICE = 0;
    public static final int ACTOR_IN_SERVICE =1;
    private Tracetrace;
    protected intactionDelayMillis;
    private AddressobservedAddress;
    private Terminal observedTerminal;
```

Actor.java

```
private boolean addressInService;
private boolean terminalInService;
protected int state = Actor.ACTOR_OUT_OF_SERVICE;

public Actor ( Trace trace, Address observed, int actionDelayMillis ) {
    this.trace = trace;
    this.observedAddress = observed;
    this.observedTerminal = observed.getTerminals ()[0];
    this.actionDelayMillis = actionDelayMillis;
}

public void initialize () {

    try {
        if ( observedAddress != null ) {
            bufPrintln (
                "Adding Call observer to address "
                + observedAddress.getName ()
                );
            observedAddress.addCallObserver ( this );

            //Now add observer on Address and Terminal
            bufPrintln (
                "Adding Adddress Observer to address "
                + observedAddress.getName ()
                );

            observedAddress.addObserver ( this );

            bufPrintln (
                "Adding Terminal Observer to Terminal"
                + observedTerminal.getName ()
                );

            observedTerminal.addObserver ( this );
        }
    }
    catch ( Exception e ) {
    } finally {
        flush ();
    }
}

public final void start () {
        onStart ();
}



public final void dispose () {

    try {
        onStop ();
        if ( observedAddress != null ) {

                bufPrintln (
                    "Removing observer from Address "
                    + observedAddress.getName ()
                    );
                observedAddress.removeObserver ( this );

                bufPrintln (
                    "Removing call observer from Address "
                    + observedAddress.getName ()
```

```
                    );
                observedAddress.removeCallObserver ( this );

            }
            if ( observedTerminal != null ){
                    bufPrintln (
                        "Removing observer from terminal "
                        + observedTerminal.getName ()
                        );
                observedTerminal.removeObserver ( this );
            }
        }
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
        finally {
            flush ();
        }
    }

    public final void stop () {
        onStop ();
    }


    public final void callChangedEvent ( CallEv [] events ) {
        //
        // for now, all metaevents are delivered in the
        // same package...
        //
        metaEvent ( events );
    }

    public void addressChangedEvent ( AddrEv [] events ) {

        for ( int i=0; i<events.length; i++ ) {
            Address address = events[i].getAddress ();
            switch ( events[i].getID () ) {
                case CiscoAddrInServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
                     addressInService = true;
                     if ( terminalInService ) {
                        if ( state != Actor.ACTOR_IN_SERVICE ) {
                            state = Actor.ACTOR_IN_SERVICE ;
                            fireStateChanged ();
                        }
                     }
                 break;
                case CiscoAddrOutOfServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for "+ address.getName ());
                     addressInService = false;
                     if ( state != Actor.ACTOR_OUT_OF_SERVICE ) {
                        state = Actor.ACTOR_OUT_OF_SERVICE; // you only want to notify when you had notified
earlier that you are IN_SERVICE
                        fireStateChanged ();
                     }
                    break;
            }
        }
        flush ();
    }

    public void terminalChangedEvent ( TermEv [] events ) {
```

Actor.java

```
        for ( int i=0; i<events.length; i++ ) {
            Terminal terminal = events[i].getTerminal ();
            switch ( events[i].getID () ) {
                case CiscoTermInServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for " + terminal.getName ());
                     terminalInService = true;
                     if ( addressInService ) {
                        if ( state != Actor.ACTOR_IN_SERVICE ) {
                            state = Actor.ACTOR_IN_SERVICE;
                            fireStateChanged ();
                        }
                     }
                     break;
                case CiscoTermOutOfServiceEv.ID:
                     bufPrintln ( "Received " + events[i] + "for " + terminal.getName () );
                     terminalInService = false;
                     if ( state != Actor.ACTOR_OUT_OF_SERVICE ) { // you only want to notify when you had
notified earlier that you are IN_SERVICE
                        state = Actor.ACTOR_OUT_OF_SERVICE;
                        fireStateChanged ();
                     }
                     break;
            }
        }
        flush();
    }

    final void delay ( String action ) {
        if ( actionDelayMillis != 0 ) {
            println ( "Pausing " + actionDelayMillis + " milliseconds before " + action );
            try {
                Thread.sleep ( actionDelayMillis );
            }
            catch ( InterruptedException e ) {}
        }
    }

    protected abstract void metaEvent ( CallEv [] events );

    protected abstract void onStart ();
    protected abstract void onStop ();
    protected abstract void fireStateChanged ();

    public final void bufPrint ( String string ) {
        trace.bufPrint ( string );
    }
    public final void bufPrintln ( String string ) {
        trace.bufPrint ( string );
        trace.bufPrint ("¥n");
    }
    public final void print ( String string ) {
        trace.print ( string );
    }
    public final void print ( char character ) {
        trace.print ( character );
    }
    public final void print ( int integer ) {
        trace.print ( integer );
    }
    public final void println ( String string ) {
        trace.println ( string );
    }
    public final void println ( char character ) {
        trace.println ( character );
```

```
    }
    public final void println ( int integer ) {
        trace.println ( integer );
    }
    public final void flush () {
        trace.flush ();
    }
}
```

Originator.java

```
/**
 * originator.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

import com.ms.com.*;
import com.cisco.jtapi.extensions.*;

public class Originator extends Actor
{
    Address srcAddress;
    String  destAddress;
    int         iteration;
    StopSignalstopSignal;
    boolean ready = false;
    int         receiverState = Actor.ACTOR_OUT_OF_SERVICE;
    boolean callInIdle = true;

    public Originator ( Address srcAddress, String destAddress, Trace trace, int actionDelayMillis ) {
        super ( trace, srcAddress, actionDelayMillis );// observe srcAddress
        this.srcAddress = srcAddress;
        this.destAddress = destAddress;
        this.iteration = 0;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnTalkingEv ) {
                    TerminalConnection tc = ((CallCtlTermConnTalkingEv)curEv).getTerminalConnection ();
                    Connection conn = tc.getConnection ();
                    if ( conn.getAddress ().getName ().equals ( destAddress ) ) {
                        delay ( "disconnecting" );
                        bufPrintln ( "Disconnecting Connection " + conn );
                        conn.disconnect ();
                    }
                }
                else if ( curEv instanceof CallCtlConnDisconnectedEv ) {
                    Connection conn = ((CallCtlConnDisconnectedEv)curEv).getConnection ();
                    if ( conn.getAddress ().equals ( srcAddress ) ) {
```

Originator.java

```
                    stopSignal.canStop ();
                    setCallProgressState ( true );
                }
            }
        }
        catch ( Exception e ) {
            println ( "Caught exception " + e );
        }
        finally {
            flush ();
        }

    }
}

protected void makecall ()
    throws ResourceUnavailableException, InvalidStateException,
          PrivilegeViolationException, MethodNotSupportedException,
          InvalidPartyException, InvalidArgumentException {
        println ( "Making call #" + ++iteration + " from " + srcAddress + " to " + destAddress + " " +
Thread.currentThread ().getName () );
        Call call = srcAddress.getProvider ().createCall ();
        call.connect ( srcAddress.getTerminals ()[0], srcAddress, destAddress );
        setCallProgressState ( false );
        println ( "Done making call" );
}


protected final void onStart () {
    stopSignal = new StopSignal ();
    new ActionThread ().start ();
}

protected final void fireStateChanged () {
    checkReadyState ();
}

protected final void onStop () {
    stopSignal.stop ();
    Connection[] connections = srcAddress.getConnections ();
    try {
        if ( connections != null ) {
            for (int i=0; i< connections.length; i++ ) {
                connections[i].disconnect ();
            }
        }
    }catch ( Exception e ) {
        println (" Caught Exception " + e);
    }
}

public int getReceiverState () {
    return receiverState;
}

public void setReceiverState ( int state ) {
    if ( receiverState != state ){
         receiverState = state;
         checkReadyState ();
    }
}
```

```
    public synchronized void checkReadyState () {
        if ( receiverState == Actor.ACTOR_IN_SERVICE && state == Actor.ACTOR_IN_SERVICE ) {
            ready = true;
        }else {
            ready = false;
        }
        notifyAll ();
    }

    public synchronized void setCallProgressState ( boolean isCallInIdle ) {
        callInIdle = isCallInIdle;
        notifyAll ();
    }

    public synchronized void doAction () {
        if ( !ready || !callInIdle ) {
            try {
                wait ();
            }catch ( Exception e ) {
                println (" Caught Exception from wait state" + e );
            }
        } else {
            if ( actionDelayMillis != 0 ) {
                println ( "Pausing " + actionDelayMillis + " milliseconds before making call " );
                flush ();
                try {
                    wait ( actionDelayMillis );
                }
                catch ( Exception ex ) {}
            }
            //make call after waking up, recheck the flags before making the call
            if ( ready && callInIdle ) {
                try {
                    makecall ();
                }catch ( Exception e ) {
                    println (" Caught Exception in MakeCall " + e + " Thread =" + Thread.currentThread
().getName ());
                }
            }
        }
    }


    class ActionThread extends Thread {

        ActionThread ( ) {
            super ( "ActionThread");
        }

        public void run () {
            while ( true ) {
                doAction ();
            }
        }
    }

}
```

Receiver.java

```
/**
 * Receiver.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.callcontrol.*;
import javax.telephony.callcontrol.events.*;

public class Receiver extends Actor
{
    Address address;
    StopSignalstopSignal;
    Originatororiginator;

    public Receiver ( Address address, Trace trace, int actionDelayMillis, Originator originator ) {
        super ( trace, address, actionDelayMillis );
        this.address = address;
        this.originator = originator;
    }

    protected final void metaEvent ( CallEv [] eventList ) {
        for ( int i = 0; i < eventList.length; i++ ) {
            TerminalConnection tc = null;
            try {
                CallEv curEv = eventList[i];

                if ( curEv instanceof CallCtlTermConnRingingEv ) {
                    tc = ((CallCtlTermConnRingingEv)curEv).getTerminalConnection ();
                    delay ( "answering" );
                    bufPrintln ( "Answering TerminalConnection " + tc );
                    tc.answer ();
                    stopSignal.canStop ();
                }
            }
            catch ( Exception e ) {
                bufPrintln ( "Caught exception " + e );
                bufPrintln ( "tc = " + tc );
            }
            finally {
                flush ();
            }
        }
    }

    protected final void onStart () {
        stopSignal = new StopSignal ();
    }

    protected final void onStop () {
        stopSignal.stop ();
        Connection[] connections = address.getConnections ();
        try {
            if ( connections != null ) {
                for (int i=0; i< connections.length; i++ ) {
                    connections[i].disconnect ();
                }
            }
```

```
        }catch ( Exception e ) {
            println (" Caught Exception " + e);
        }
    }

    protected final void fireStateChanged () {
        originator.setReceiverState ( state );
    }
}
```

StopSignal.java

```
/**
 * StopSignal.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */

class StopSignal {
    boolean stopping = false;
    boolean stopped = false;
    synchronized boolean isStopped () {
        return stopped;
    }
    synchronized boolean isStopping () {
        return stopping;
    }
    synchronized void stop () {
        if ( !stopped ) {
            stopping = true;
            try {
                wait ();
            }
            catch ( InterruptedException e ) {}
        }
    }
    synchronized void canStop () {
        if ( stopping = true ) {
            stopping = false;
            stopped = true;
            notify ();
        }
    }
}
```

Trace.java

```
/**
 * Trace.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */
public interface Trace
{
```

```
/**
     * bufPrint (str) puts str in buffer only.
     */
    public void bufPrint ( String string );

    /**
     * print () println () bufPrint and invoke flush ();
     */
    public void print ( String string );
    public void print ( char character );
    public void print ( int integer );
    public void println ( String string );
    public void println ( char character );
    public void println ( int integer );

    /**
     * flush out the buffer.
     */
    public void flush ();
}
```

TraceWindow.java

```
/**
 * TraceWindow.java
 *
 * Copyright Cisco Systems, Inc.
 *
 */


import java.awt.*;
import java.awt.event.*;

public class TraceWindow extends Frame implements Trace
{

    TextArea textArea;
    booleantraceEnabled = true;
    StringBuffer buffer = new StringBuffer ();

    public TraceWindow (String name ) {
        super ( name );
        initWindow ();
    }

    public TraceWindow(){
        this("");
    }


    private void initWindow() {
        this.addWindowListener(new WindowAdapter () {
            public void windowClosing(WindowEvent e){
                dispose ();
            }
        });
        textArea = new TextArea();
        setSize(400,400);
        add(textArea);
```

```
        setEnabled(true);
        this.show();

    }


    public final void bufPrint ( String str ) {
        if ( traceEnabled ) {
            buffer.append ( str );
        }

    }

    public final void print ( String str ) {
        if ( traceEnabled ) {
            buffer.append ( str );
            flush ();
        }
}
public final void print ( char character ) {
        if ( traceEnabled ) {
            buffer.append ( character );
            flush ();
        }
}
public final void print ( int integer ) {
        if ( traceEnabled ) {
            buffer.append ( integer );
            flush ();
        }
}
    public final void println ( String str ) {
        if ( traceEnabled ) {
            print ( str );
            print ( '¥n' );
            flush ();
        }
}
public final void println ( char character ) {
        if ( traceEnabled ) {
            print ( character );
            print ( '¥n' );
            flush ();
        }
}
public final void println ( int integer ) {
        if ( traceEnabled ) {
            print ( integer );
            print ( '¥n' );
            flush ();
        }
}

public final void setTrace ( boolean traceEnabled ) {
    this.traceEnabled = traceEnabled;
}

    public final void flush () {
        if ( traceEnabled ) {
            textArea.append ( buffer.toString());
            buffer = new StringBuffer ();
        }
    }
```

```
  public final void clear () {

    textArea.setText("");
  }
}
```

makecall の実行

クライアント ワークステーション上で Windows のコマンド ラインから makecall を起動するには、JTAPI Tools ディレクトリをインストールした場所にある **makecall** ディレクトリに移動し、次のコマンドを実行します。

```
jview makecall <server name> <login> <password> 1000 <device 1> <device2>
```

<server name> には Cisco Unified Communications Manager のホスト名または IP アドレスを指定し、<device1> <device2> には IP フォンの電話番号を指定します。この IP フォンは、Cisco Unified Communications Manager のディレクトリ管理で管理されている任意のユーザの関連デバイスの一部である必要があります。<login> および <password> にも、同様にディレクトリで管理されているものを指定します。これにより、インストールと構成が正しく行われていることをテストできます。このアプリケーションは、停止されるまでの間、1000 ミリ秒の動作遅延で 2 台のデバイス間において発呼します。

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。

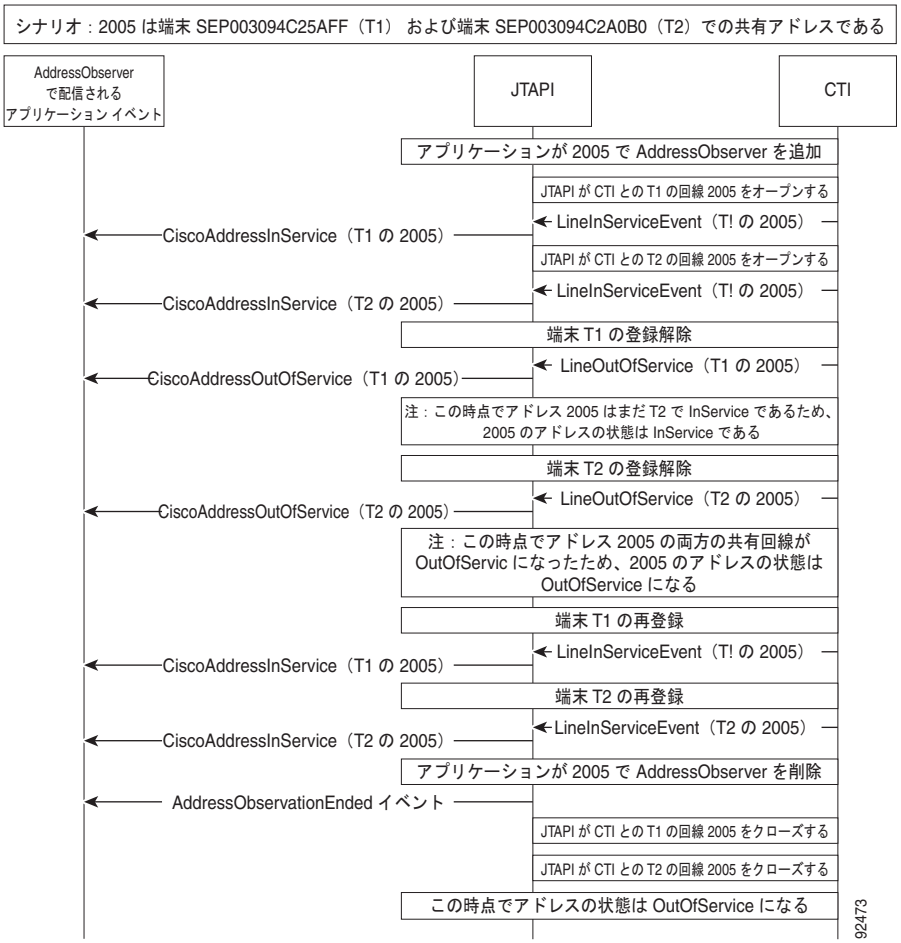
- 共用回線のサポート
- 転送と直接転送
- 会議と参加
- 割り込みとプライバシー
- CallSelect と UnSelect
- コールごとの CTIPort 動的登録
- ルート ポイントでのメディア終端
- リダイレクト時のオリジナルの着信者番号
- シングル ステップ転送
- 発信側番号の変更
- CTIPort および RoutePoint での AutoAccept
- Forced Authorization Code と Customer Matter Code
- スーパー プロバイダーのメッセージ フロー
- スーパープロバイダーと変更通知の拡張の使用例
- QSIG 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

- 録音と監視
- インターコム
- Do Not Disturb（サイレント）
- DND–R
- セキュア会議
- JTAPI Cisco Unified IP 7931G Phone の対話
- ロケール インフラストラクチャ変更シナリオ
- 発信側の正規化
- クリック ツー会議
- コール ピックアップ
- 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()
- エクステンション モビリティ ログイン ユーザ名
- 発信側の IP アドレス
- CiscoJtapiProperties
- IPv6 のサポート
- 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
- 拡張された MWI の使用例
- 回線をまたいで参加（Join Across Lines）の拡張機能
- スワップ/キャンセルおよび転送/会議の動作変更
- 任意の通話者のドロップ（Drop Any Party）の使用例
- 「パーク モニタリング サポート」（P.A-261）
- 論理パーティション設定機能の使用例
- ComponentUpdater 拡張の使用例
- IPv6 のサポート
- 「Cisco Unified IP Phone 6900 シリーズのサポート」（P.A-297）

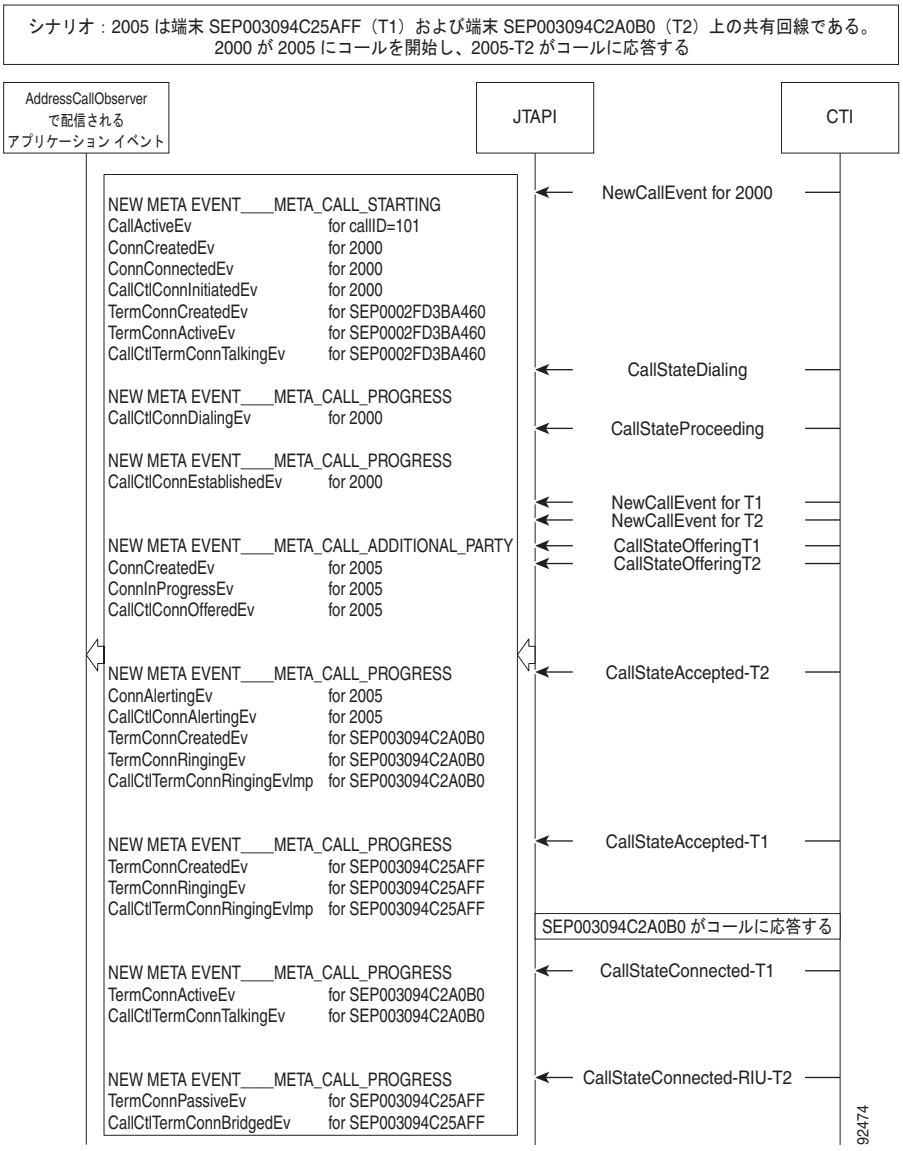
共用回線のサポート

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。

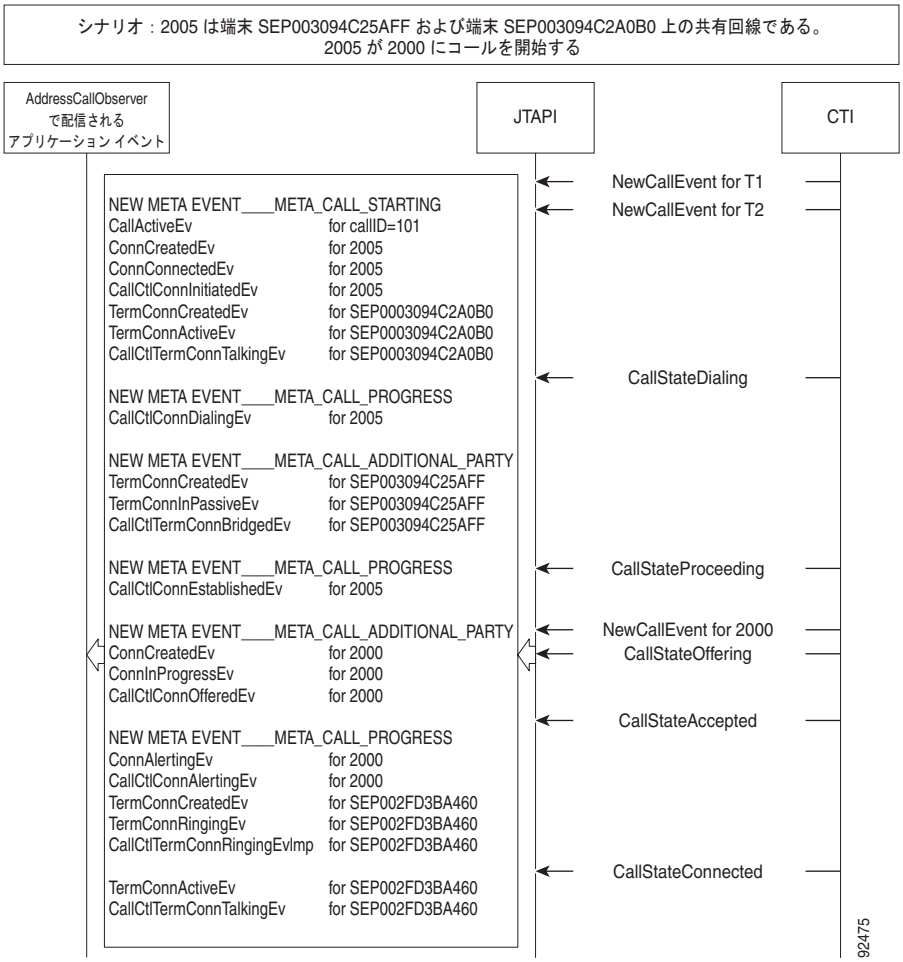
AddressInService/AddressOutofService イベント



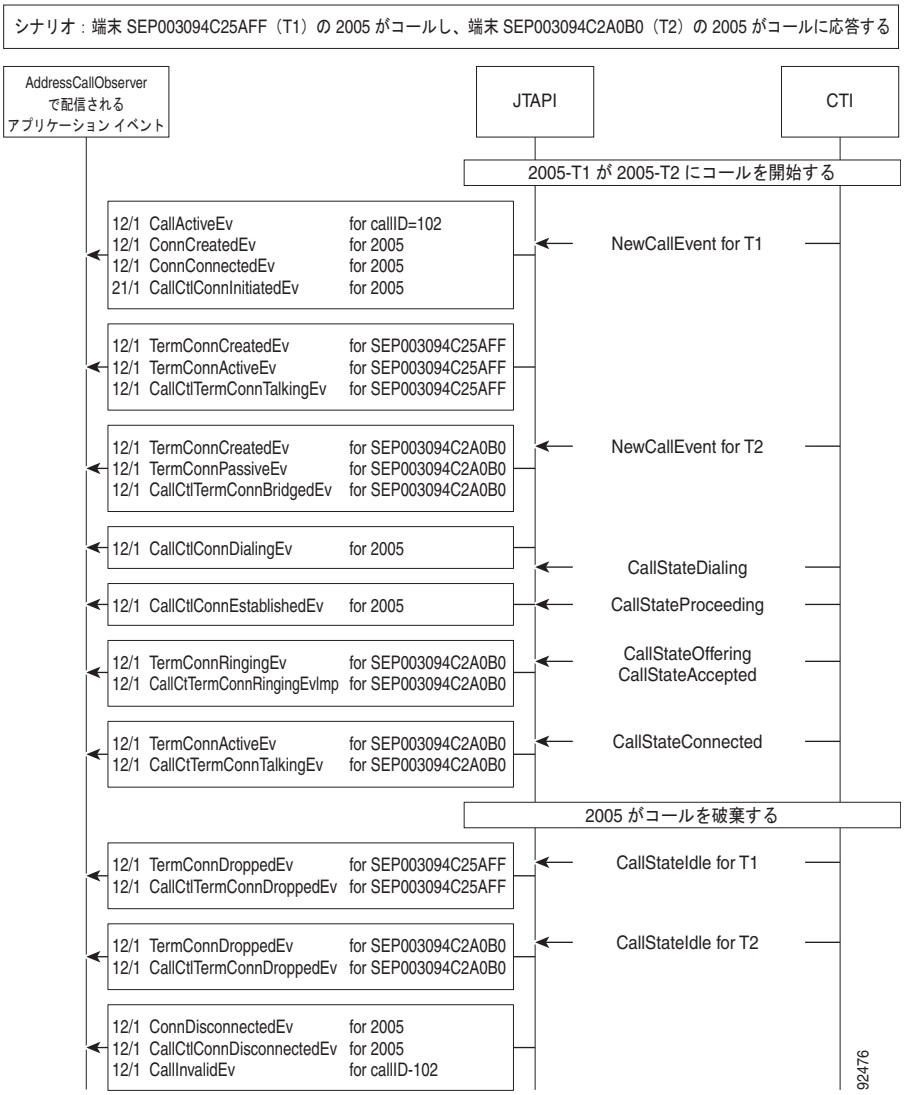
共用アドレスへの着信コール



共用アドレスからの発信コール



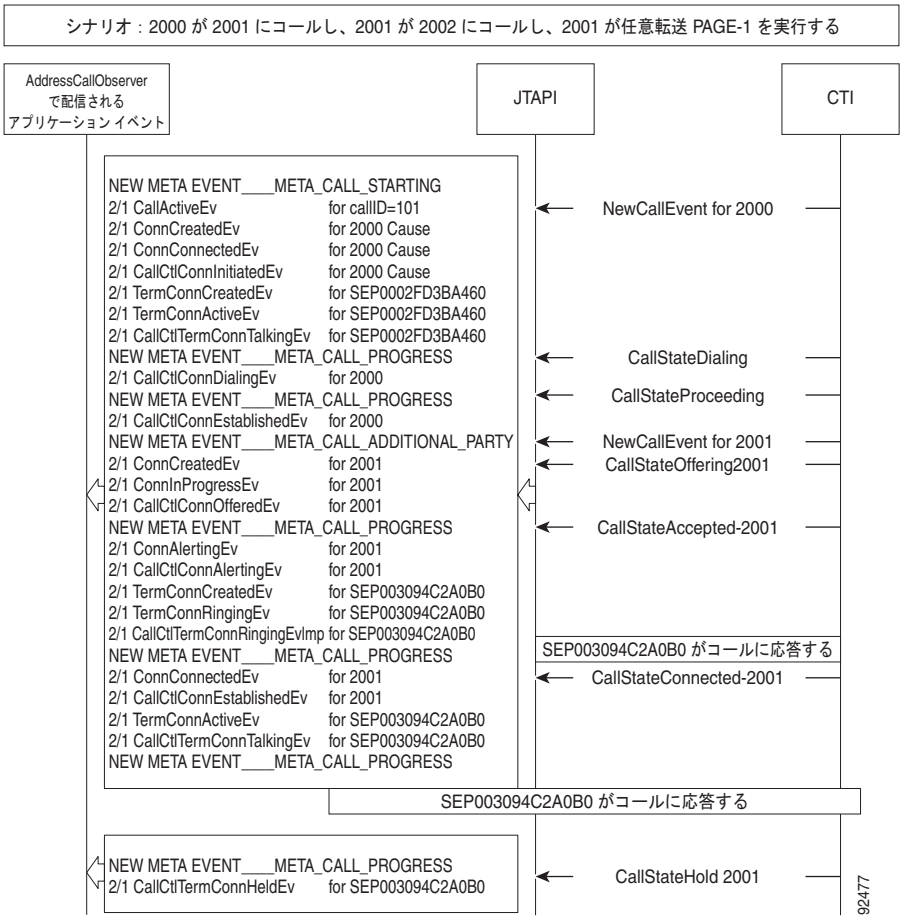
共用アドレス自体へのコール



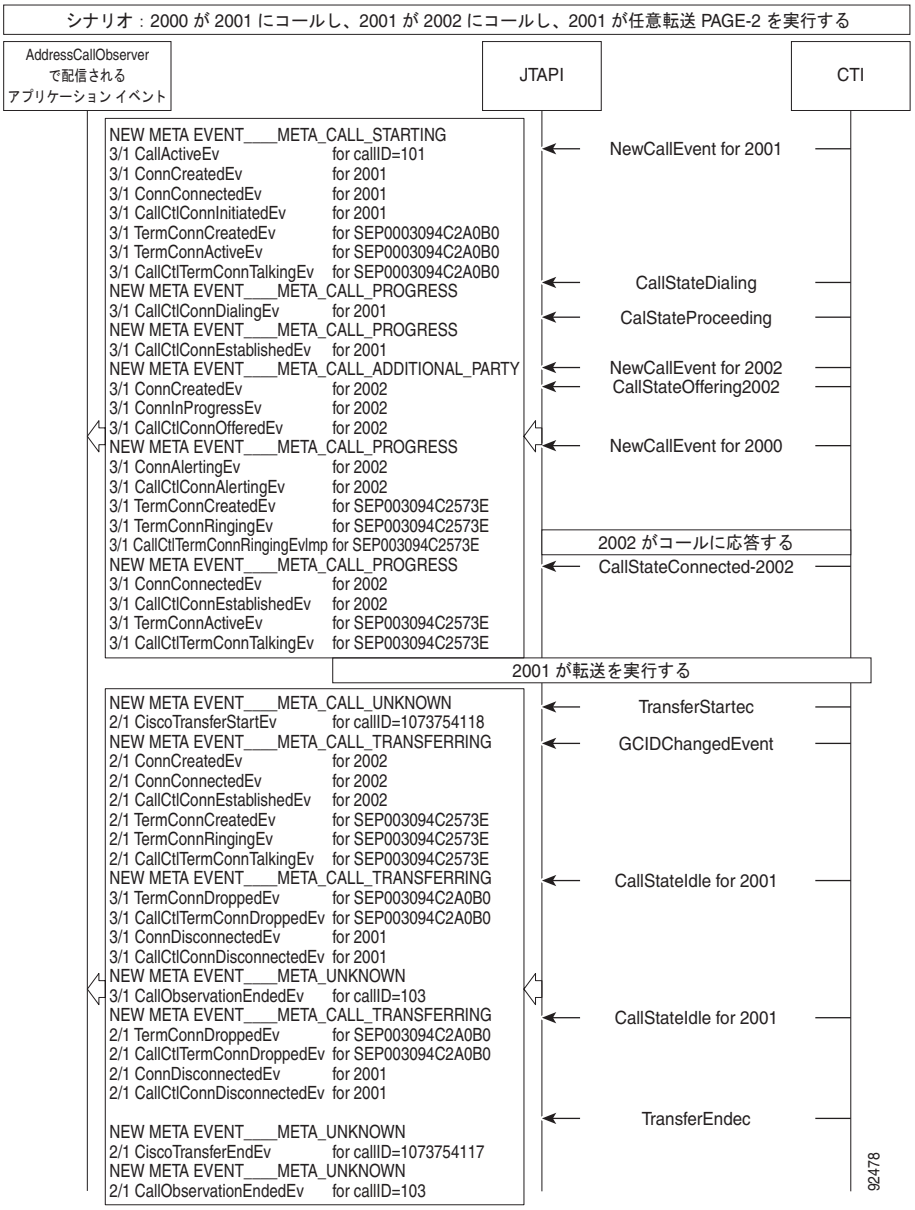
転送と直接転送

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



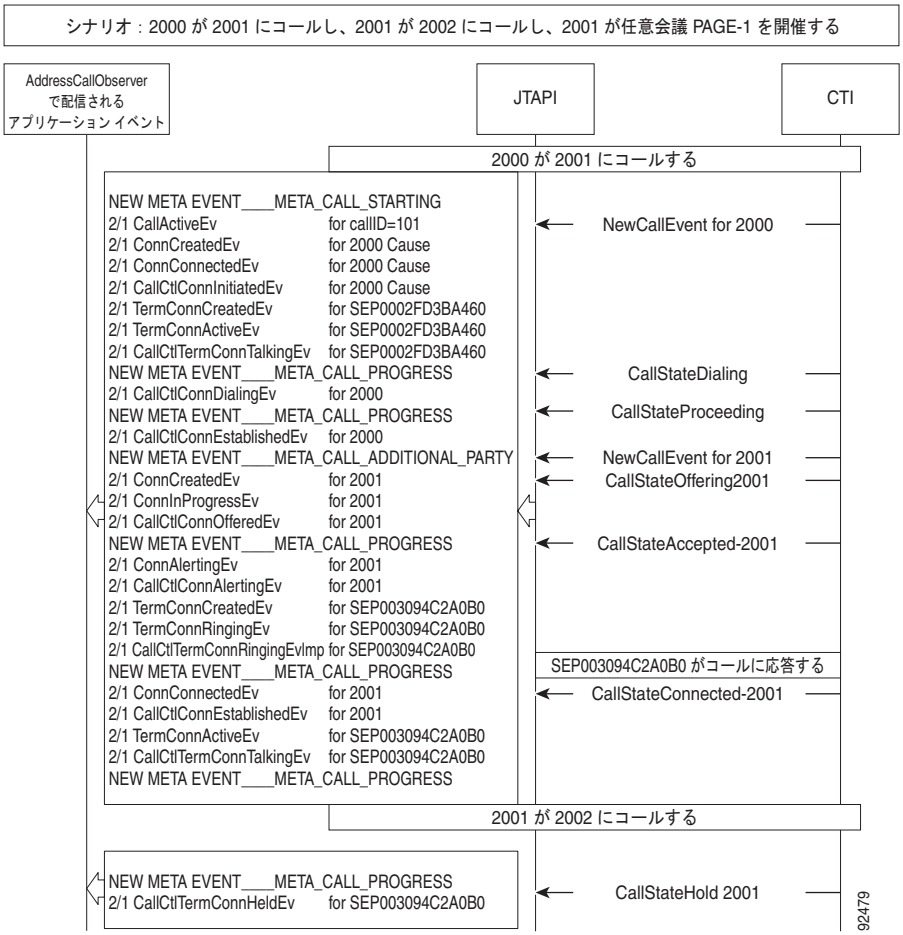
コンサルト転送

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。

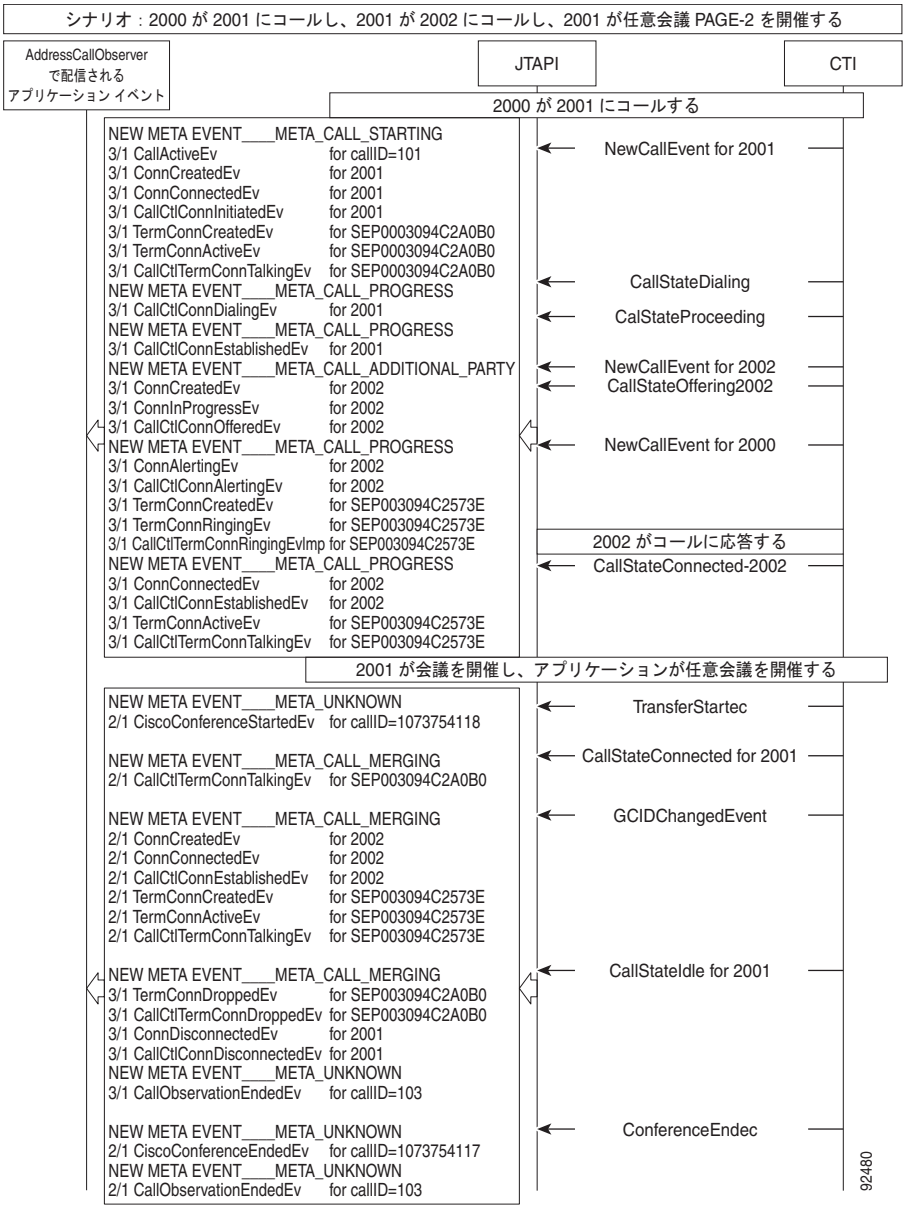
会議と参加

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。

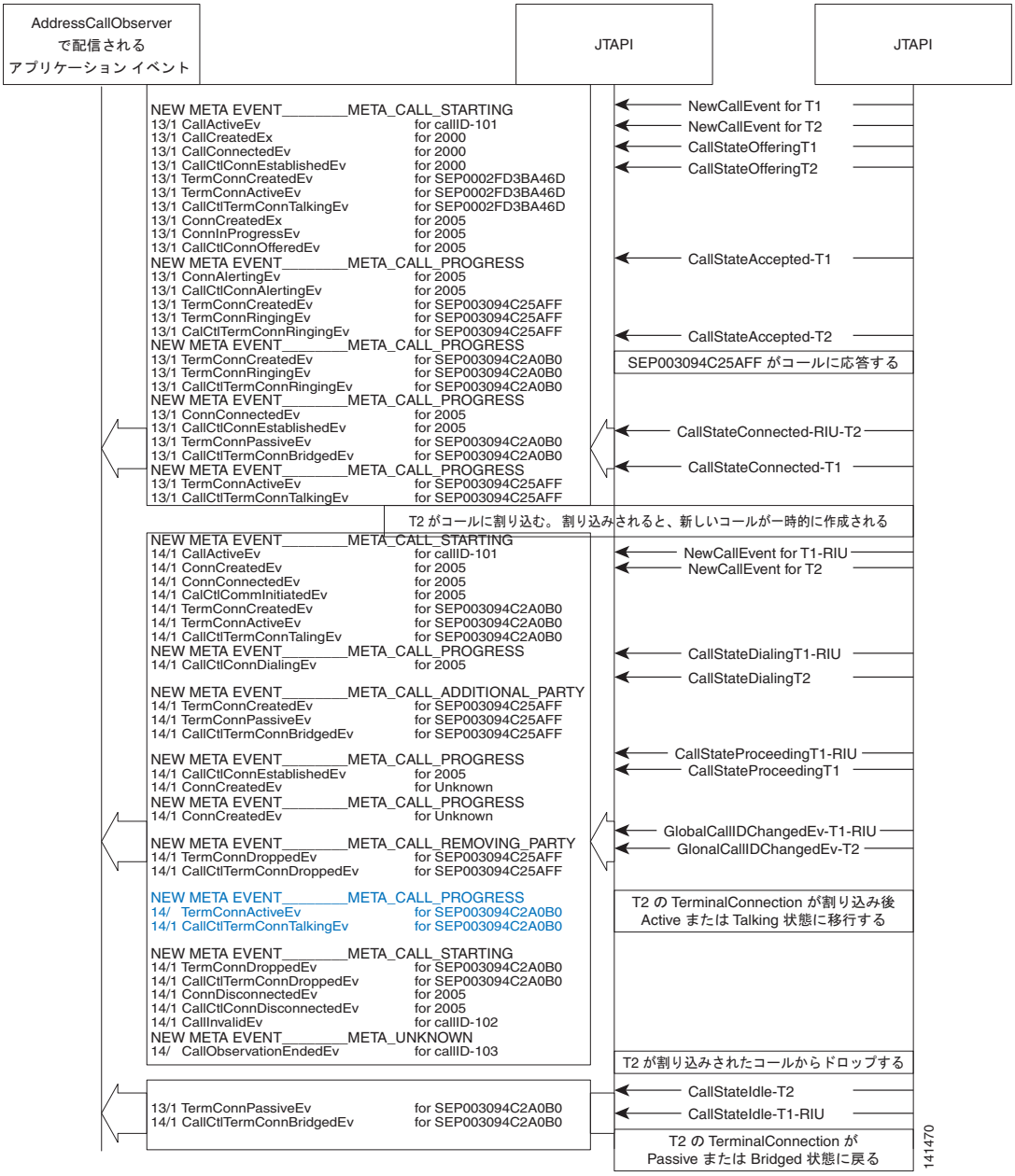
操作	イベント
アプリケーションはリクエスタを B2 と設定し、GC2.conference(GC1) を呼び出し、getControllerAddress() は B2 を返す。 getOriginalControllerAddress() は B1 を返す。	A CiscoConferenceStartEv CallCtlTermConnTalkingEv TermB GC1 ConnCreatedEv D GC1 ConnConnectedEv D GC1 CallCtlTermConnDroppedEv TermB GC2 CiscoConferenceEndEv B1 CallCtlTermConnHeldEv 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。

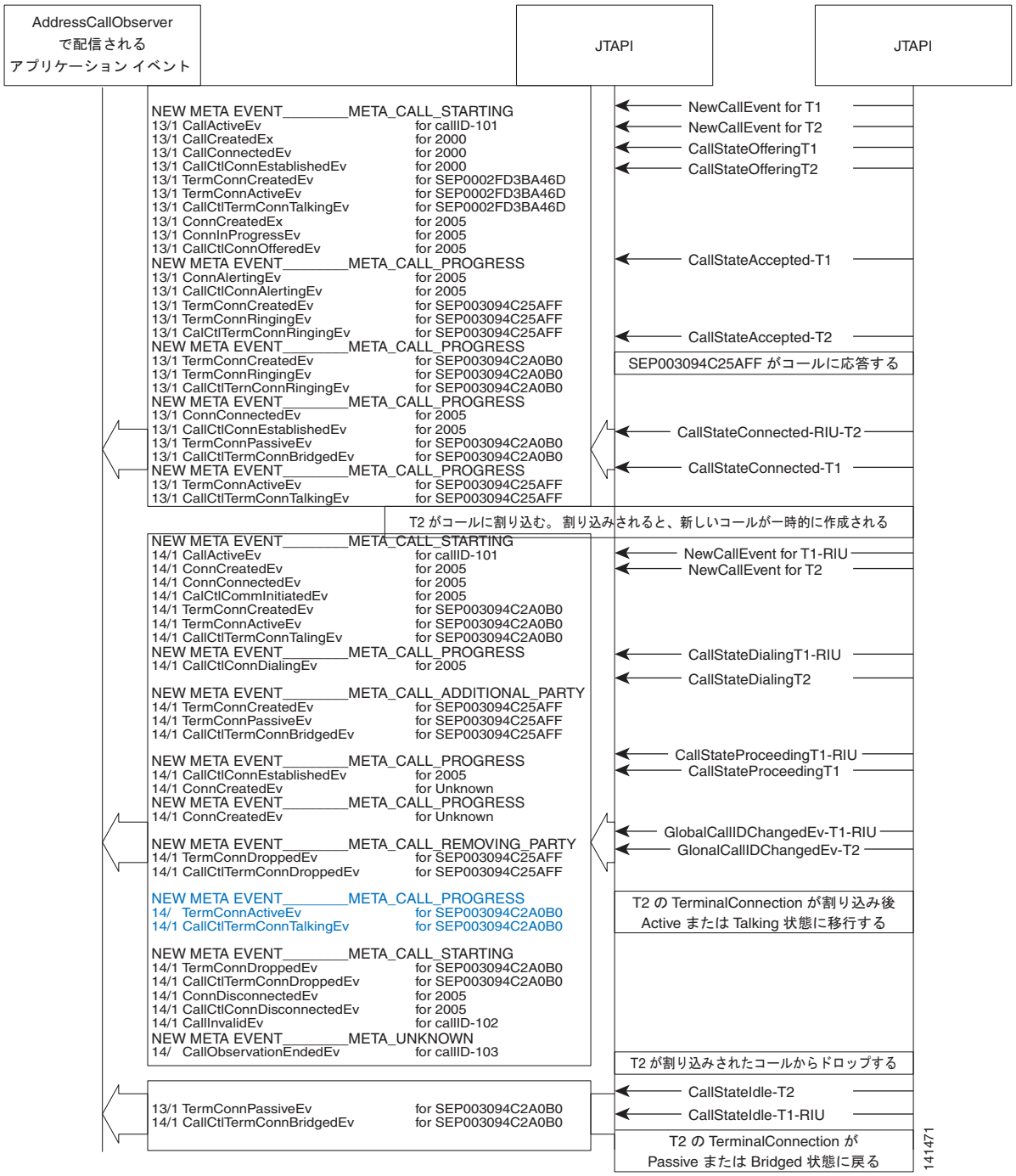
割り込み

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

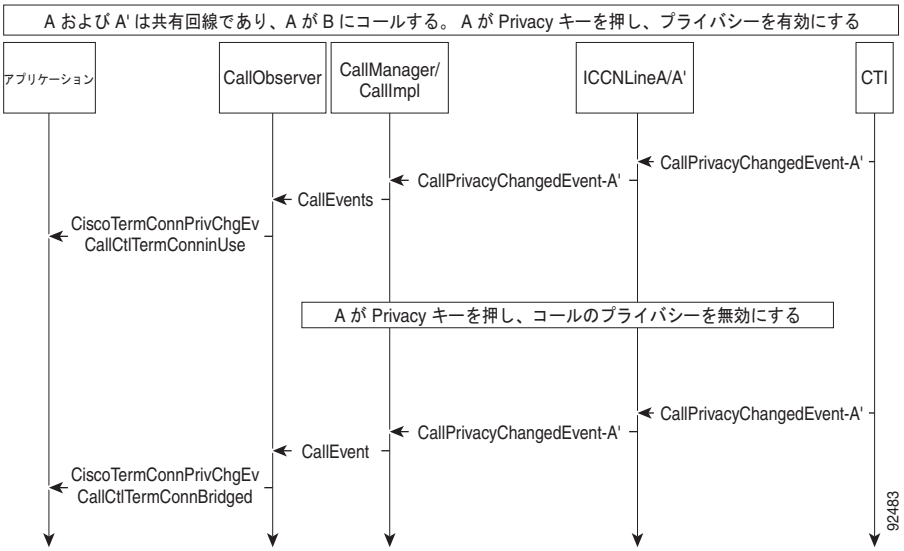


C 割込

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

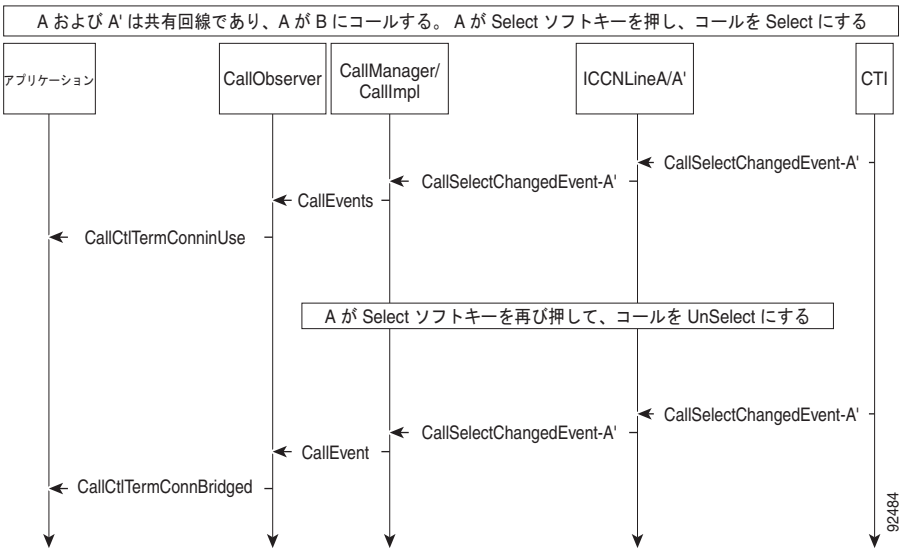


プライバシー



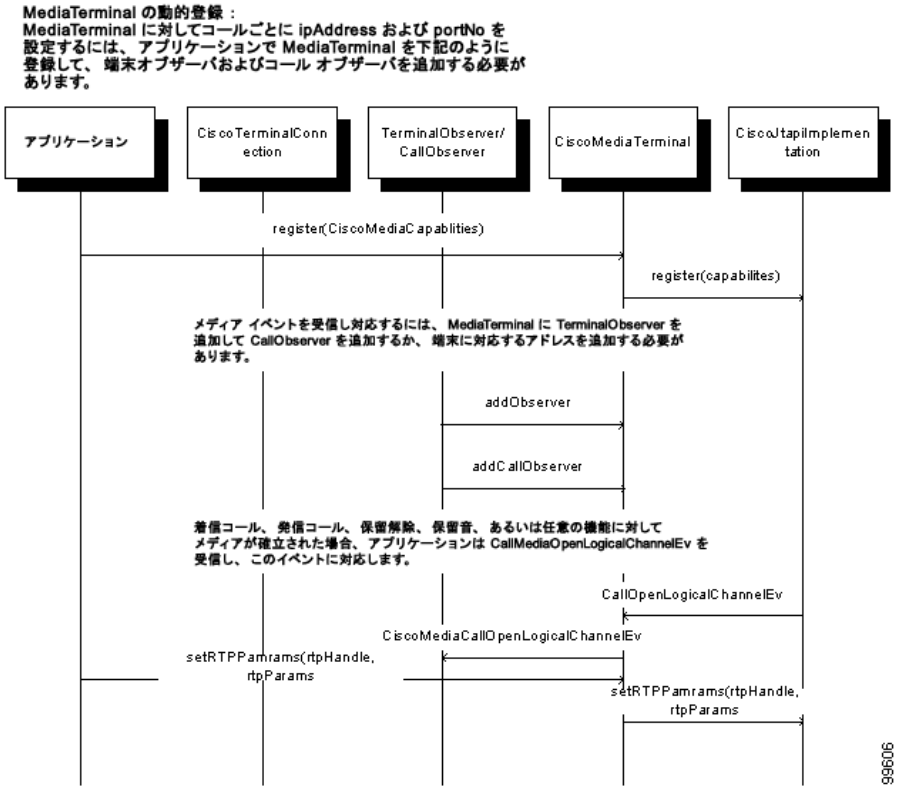
CallSelect と UnSelect

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

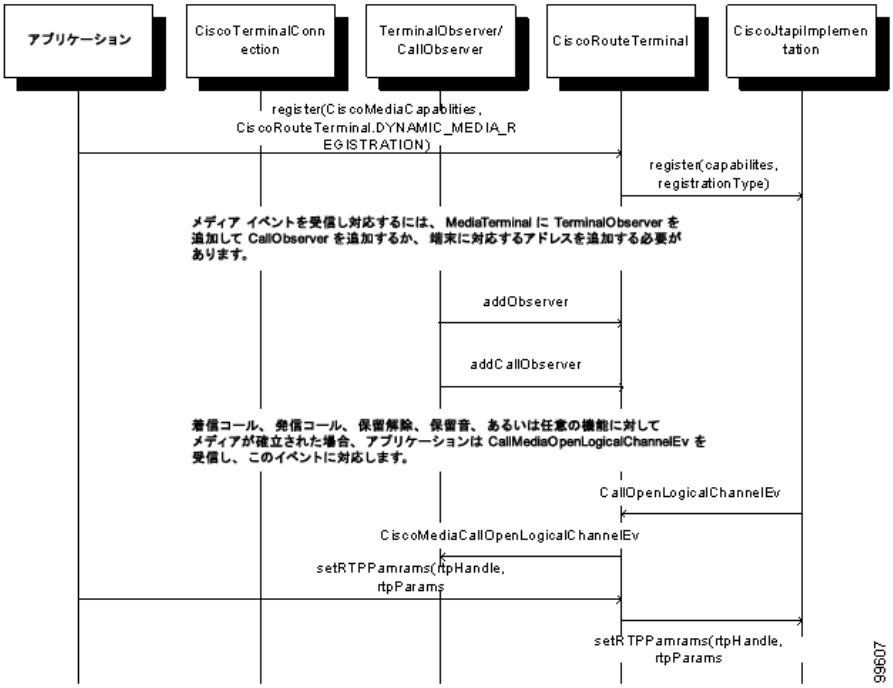
コールごとの CTIPort 動的登録に関するメッセージ フローを次に示します。



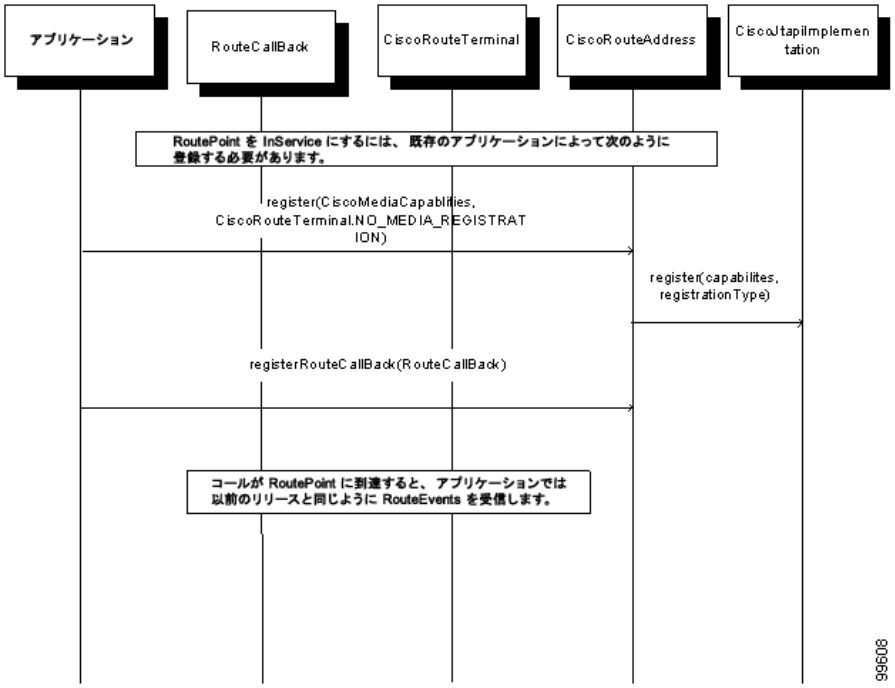
ルート ポイントでのメディア終端

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

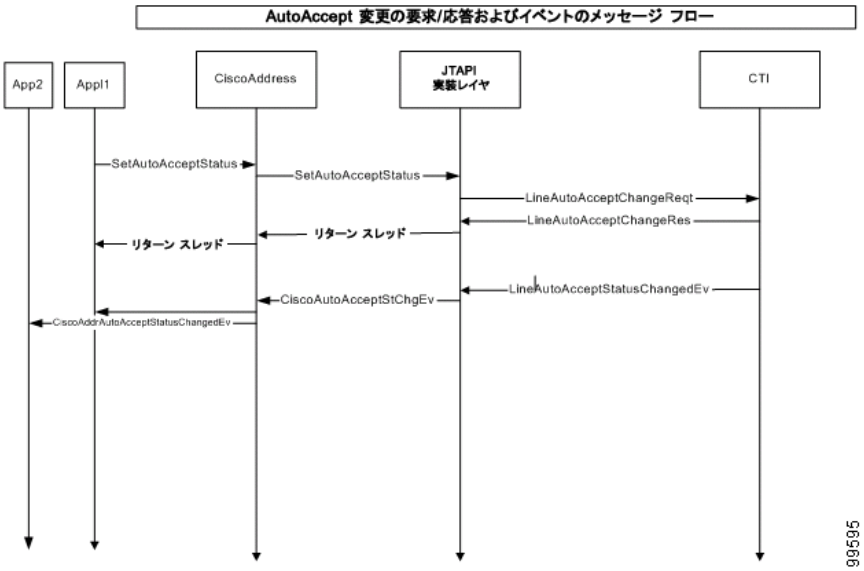
アプリケーションが A と B を制御している。

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

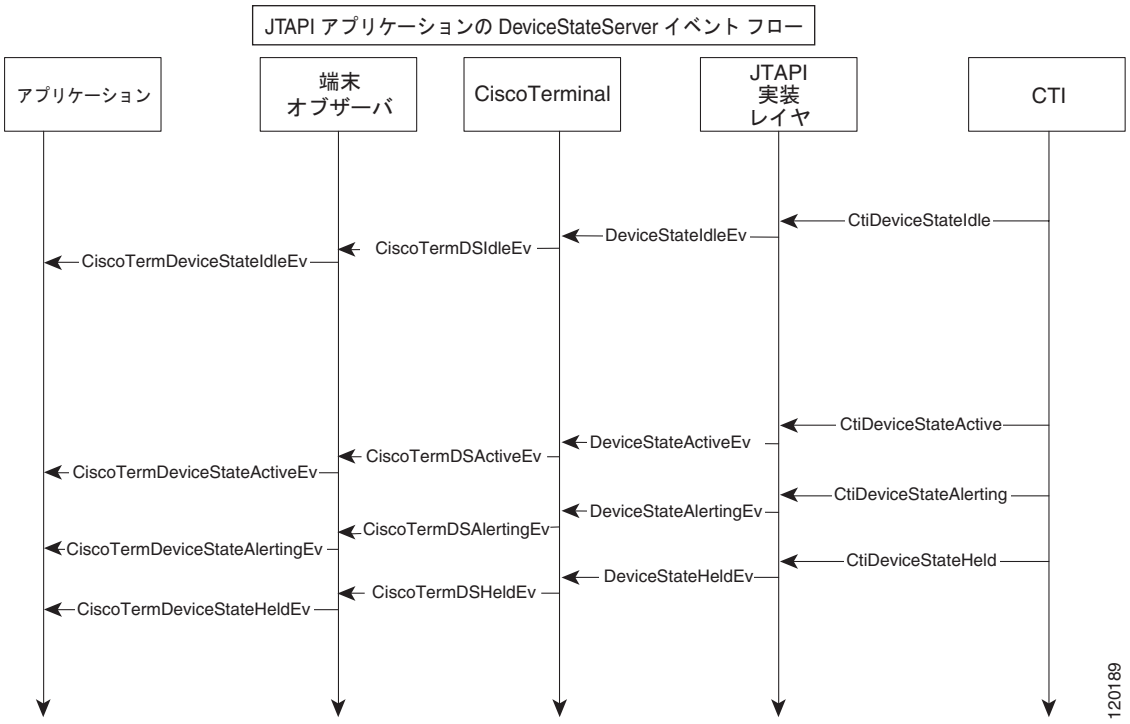
次の表は、Q.Signaling（QSIG）トランクで接続された PBX 間のコールが転送されたときに、アプリケーションに送信される JTAPI イベントを示しています。また、QSIG パス置換機能によってリアルタイム パス（RTP）が最適化されたときにアプリケーションに送信されるイベントも示しています。

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



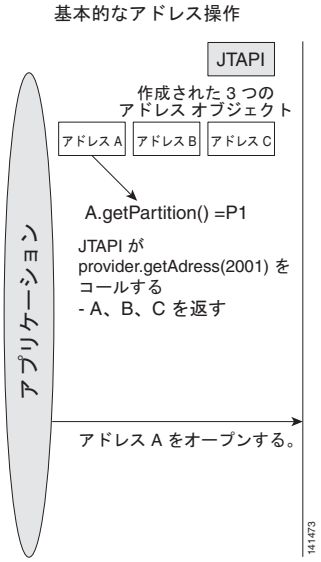
パーティションのサポート

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。

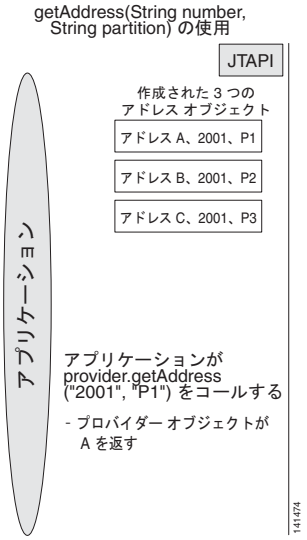
JTAPI が provider.getAddress("2001") をコールすると、プロバイダー オブジェクトは A、B、および C の 3 つのアドレス オブジェクトを含む配列を返します。これはこの 3 つがすべて同じ DN を持っているためです。

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。

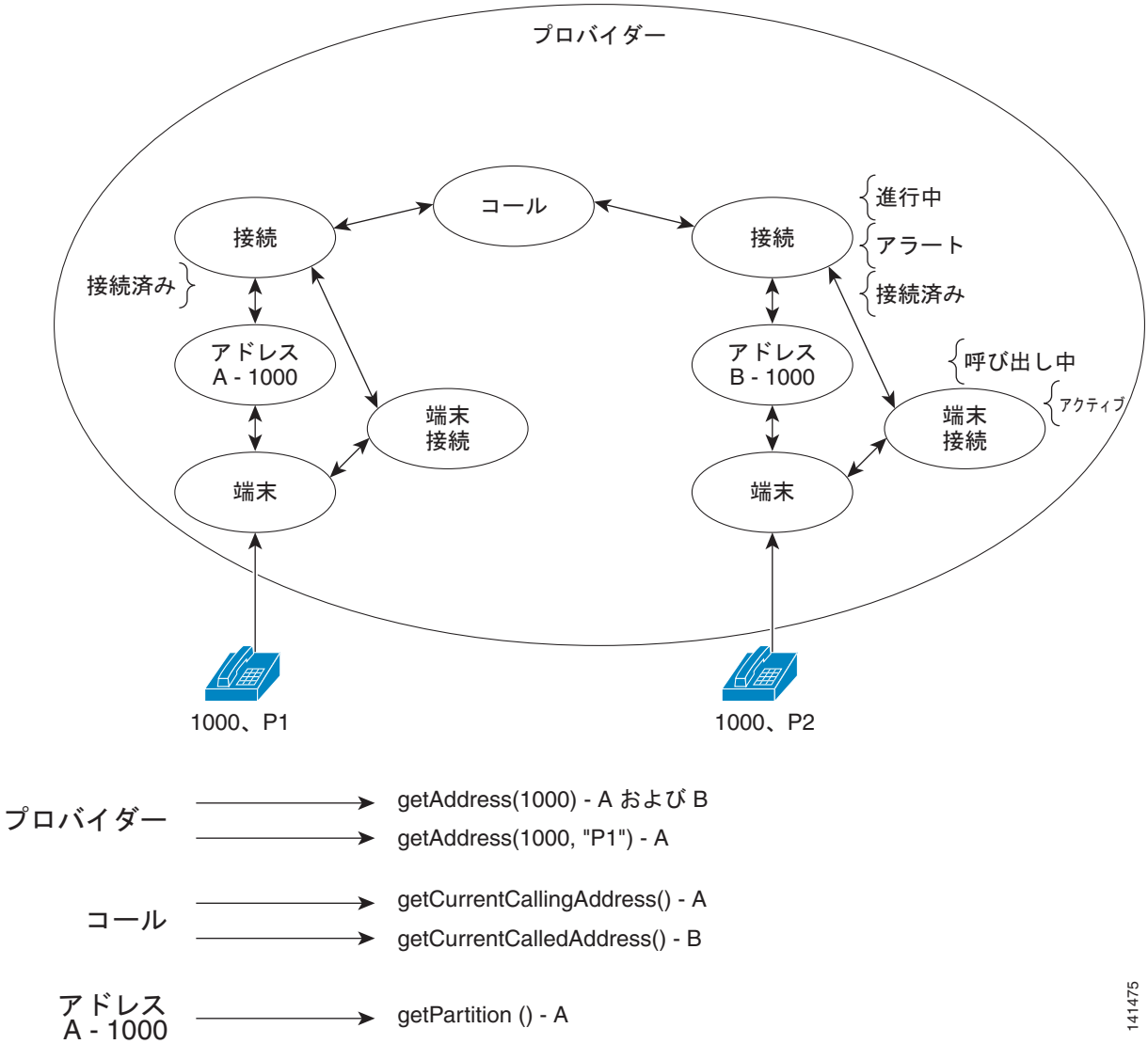
JTAPI が ***provider.getAddress("2001", "P1")*** をコールすると、プロバイダー オブジェクトは API で指定したのと同じ DN（2001）とパーティション情報（P1）を持つアドレス オブジェクトを返します。この場合、アプリケーションにはアドレス オブジェクト A が返されます。

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

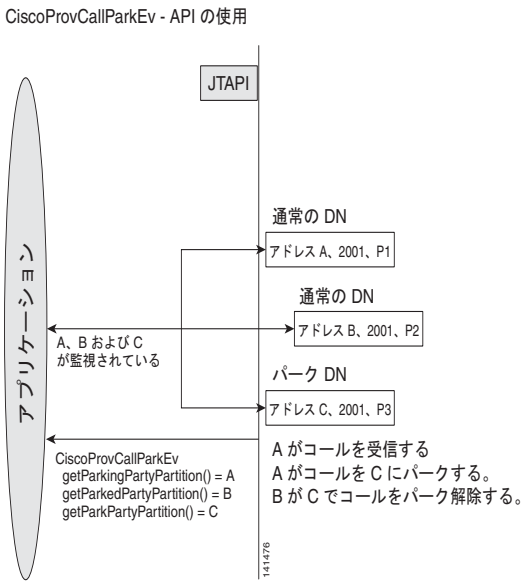
シナリオ：回線 x1000 :P1" から 回線 x1000 "P2" へのコール



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。

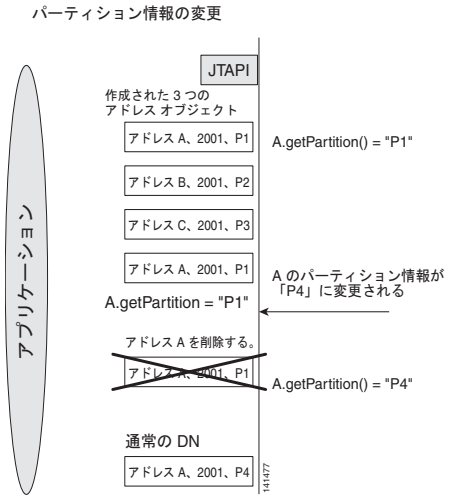
A がコールを受信してそれを C にパークします。B がそのコールをパーク解除します。コールがパークおよびパーク解除されると、CiscoProvCallParkEv が生成されます。API

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。

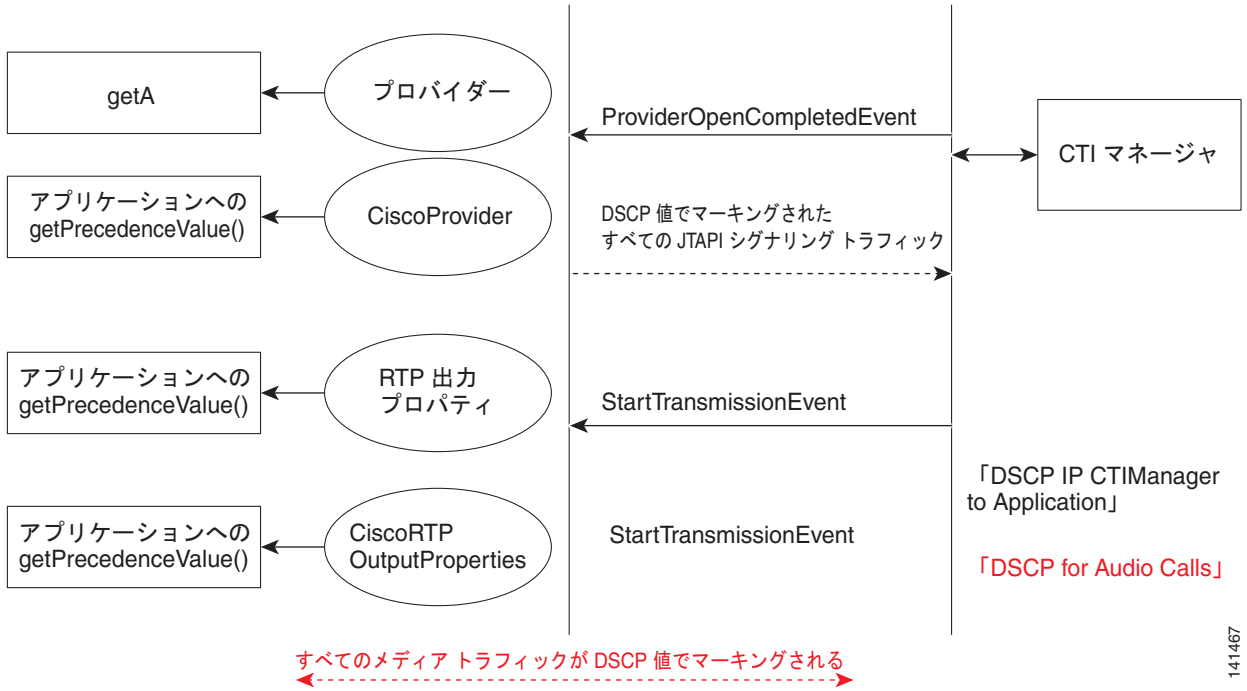
ヘアピン サポート

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

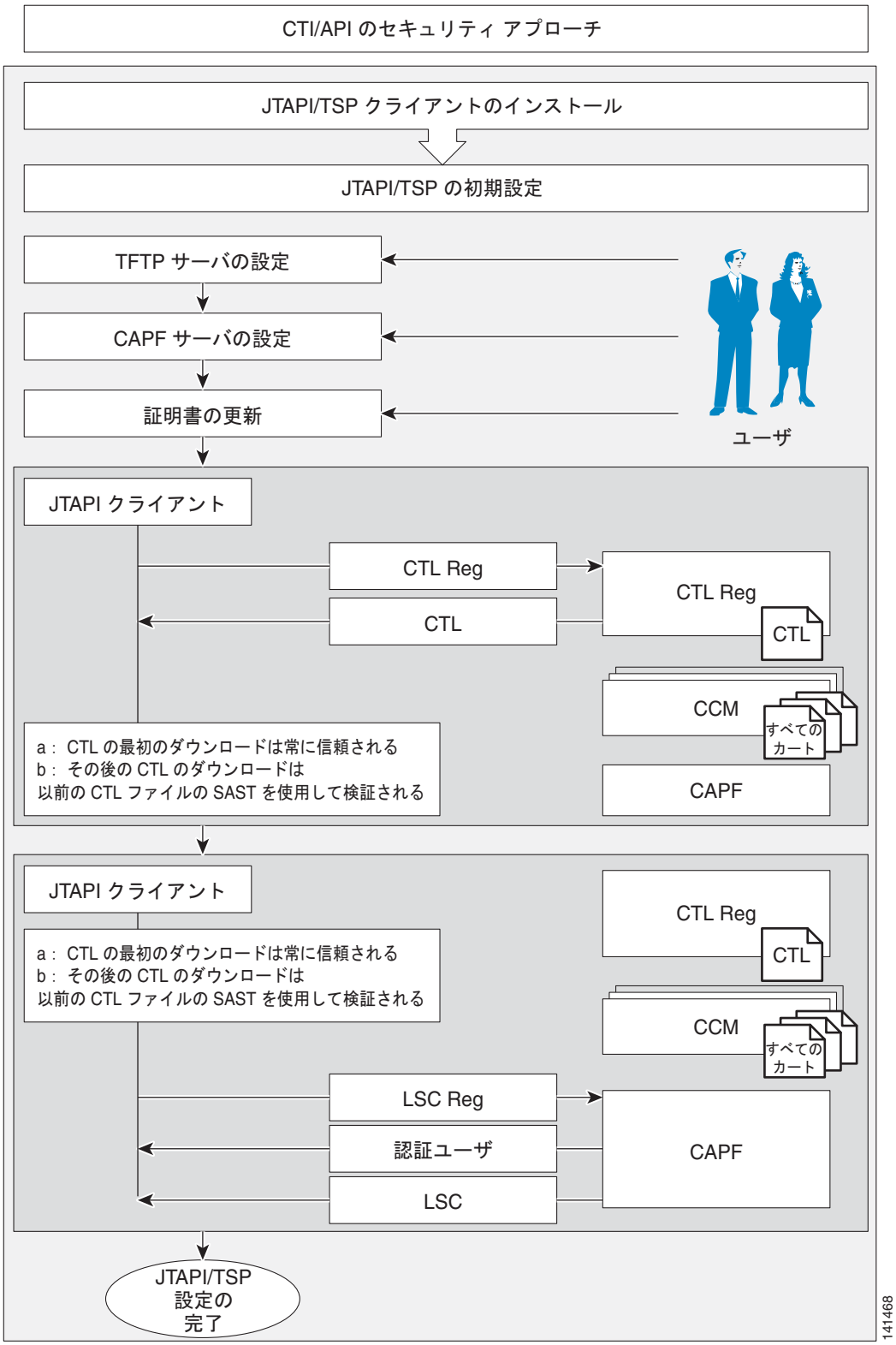
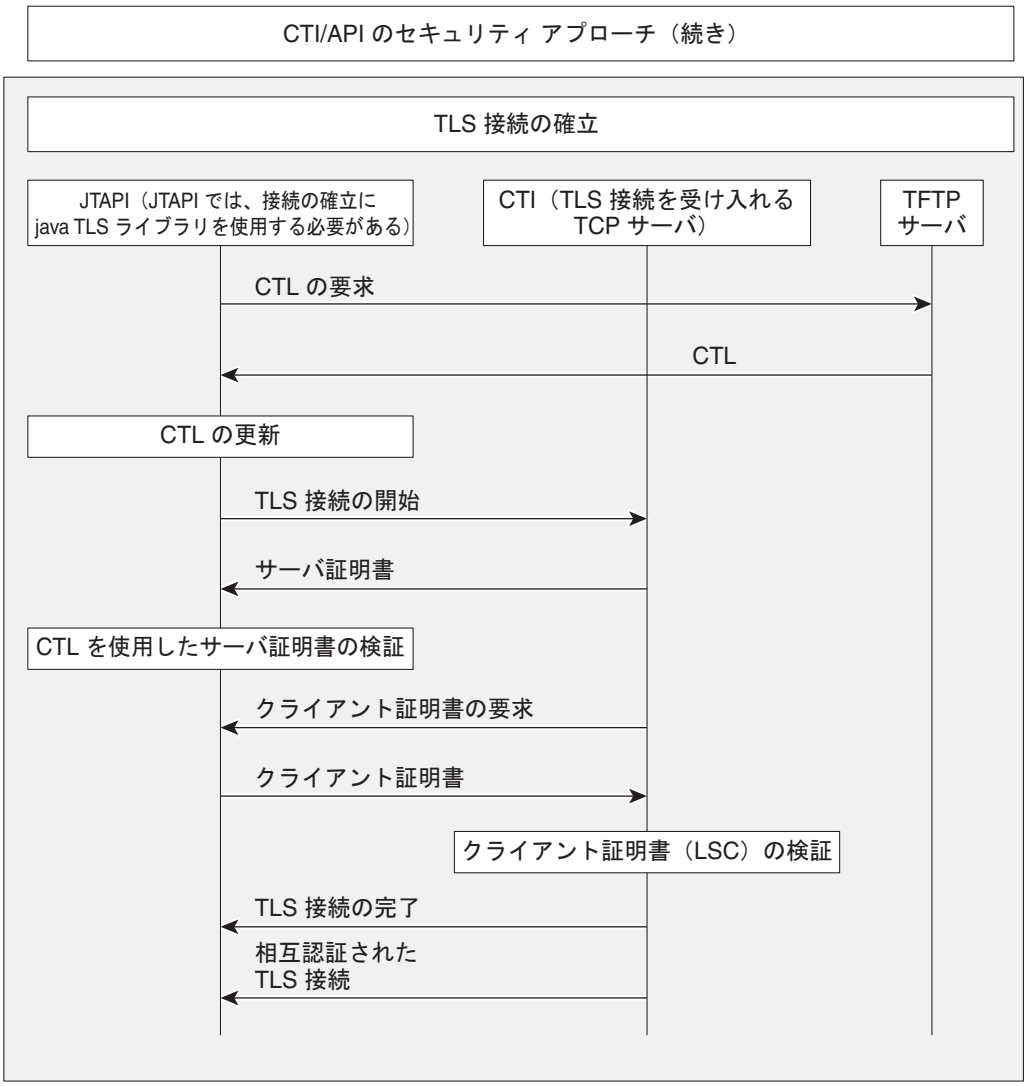


図 A-3　CTI/API のセキュリティ アプローチ（続き）



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 には CiscoAddrRestrictedEv A1 には CiscoAddrOutOfServiceEv
	A1 は通話者 X とのコールに参加する。	ConnDisconnectedEv CallCtlConnDisconnectedEv TermConnDroppedEv
	A1 が制限リストに追加される。	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 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 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 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、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

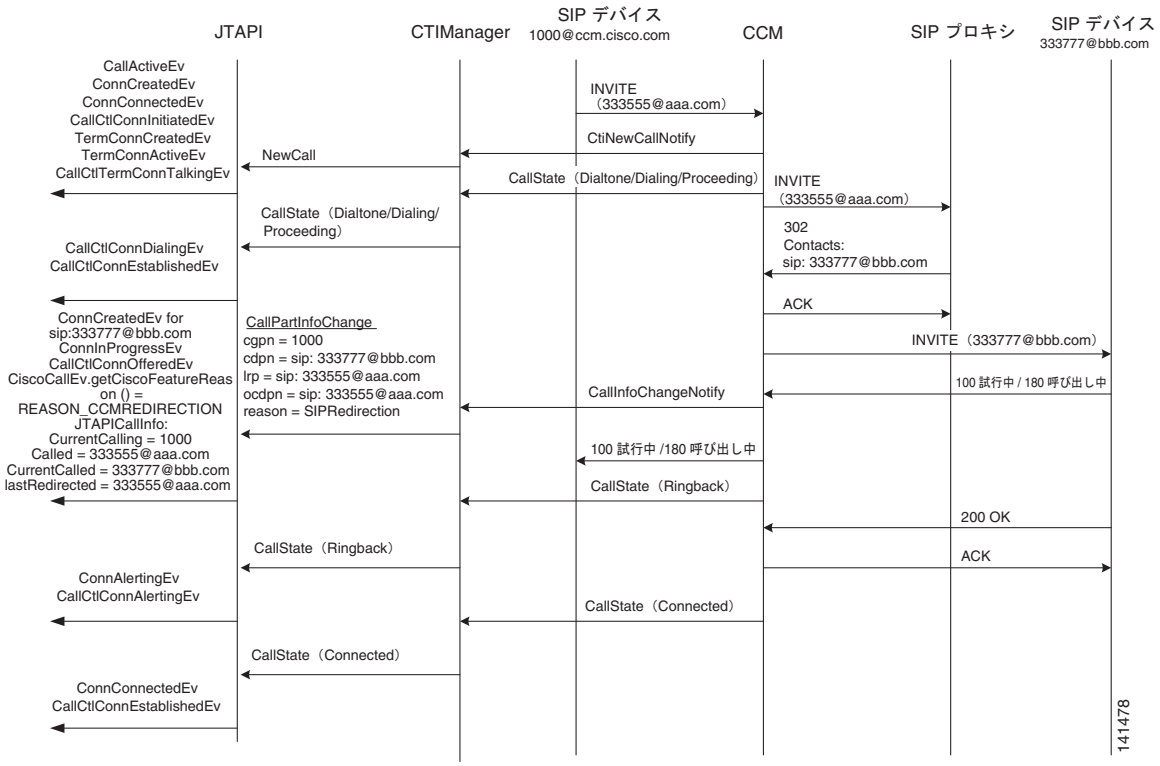
getCurrentCallingParty()=B

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。

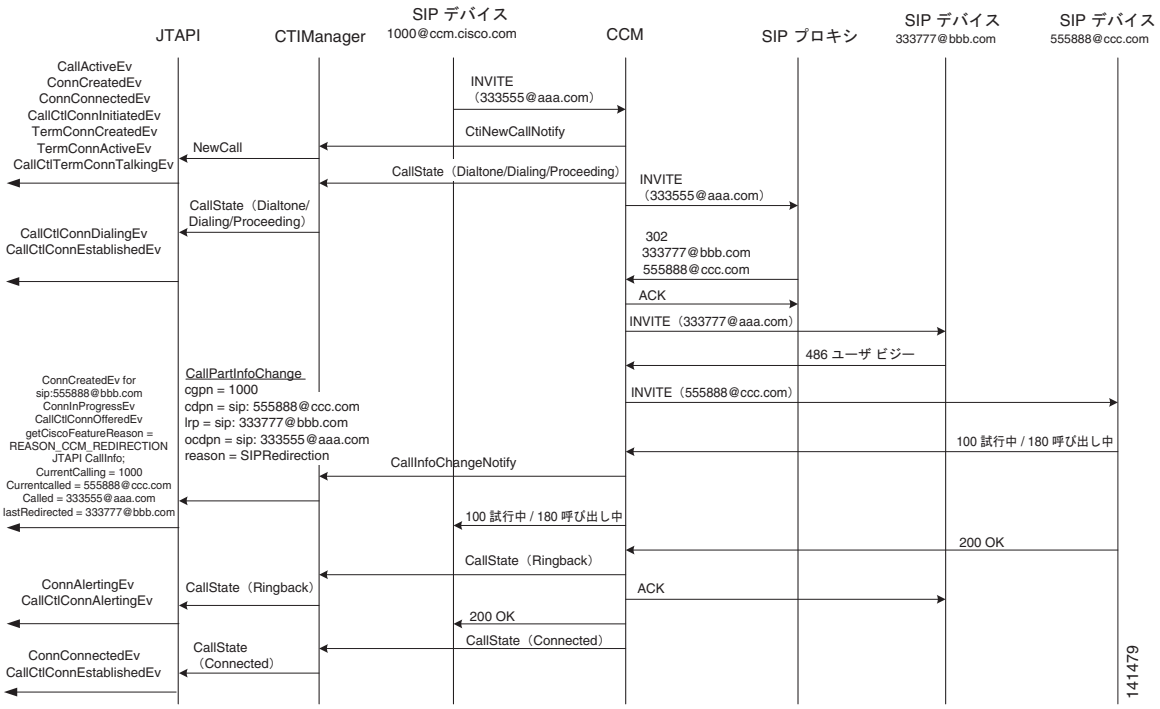
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

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。

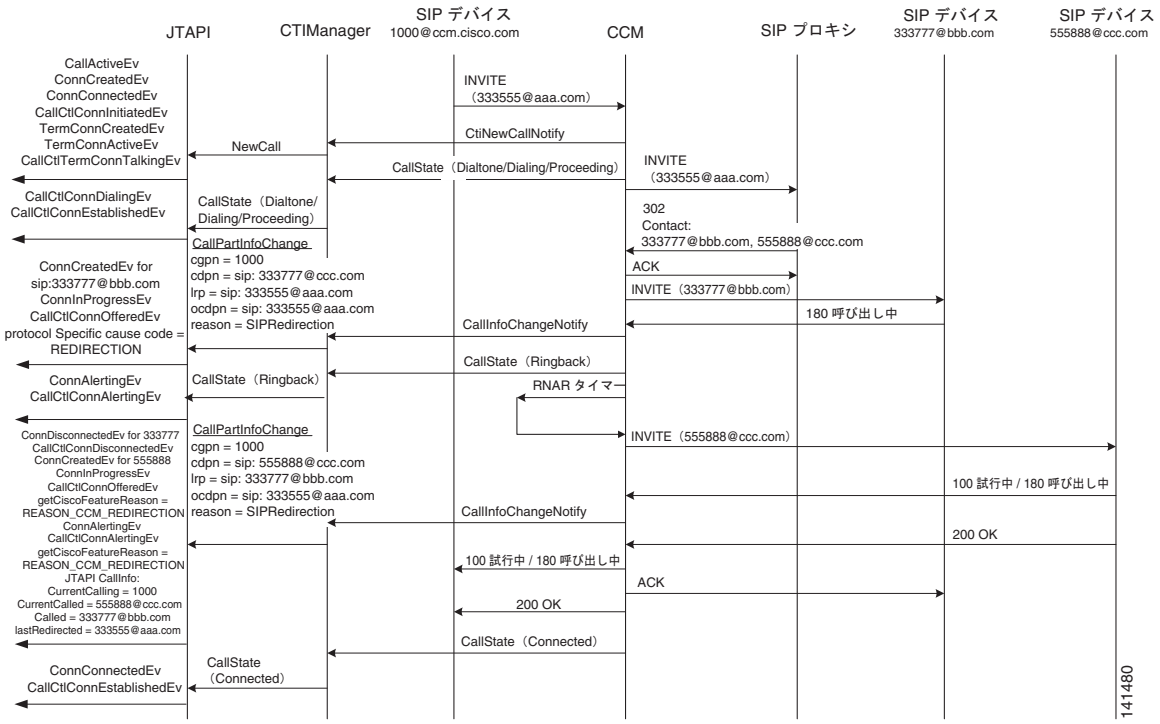
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

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。

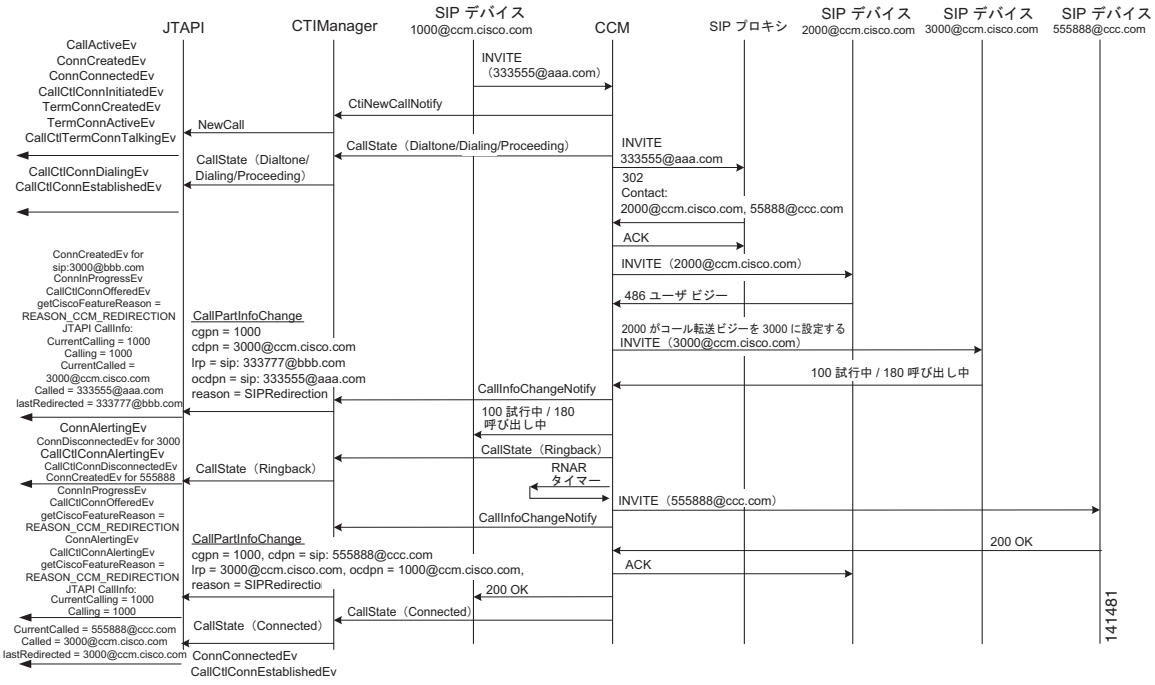
着信者情報が変更された場合、Cisco Unified Communications Manager からの SIPAlertInd/CcNotifyReq に基づいてアプリケーションに CallPartyInfoChange イベントが報告される。

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	Current called: A
	CallCrlTermConnTalkingEv TA Cause: CAUSE_NORMAL	
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	
	CiscoTermConnRecordingStartEv Cause: CAUSE_NORMAL	
A が、コールを再開する。	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
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: CallCtlTermConnBridgedEv TermA'	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A
A' はコールに割込む。	GC1: **GC1:CallCrlTermConnTalkingEv** TA'	
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。	startMonitor 要求に例外がスローされる。	

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' **GC1: CiscoTermConnMonitorInitiatorInfoEv TA'** Cause: CAUSE_NORMAL address:B, device name: TB	
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。		

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:ConnCreatedEv** for A Cause: CAUSE_SNAPSHOT **GC1:ConnConnectedEv** for A Cause: CAUSE_SNAPSHOT GC1: ConnConnectedEv X ... **GC1:CallCrlTermConnTalkingEv** TA Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnRecordingTargetInfoEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorStartEv TA** Cause: CAUSE_SNAPSHOT **GC1: CiscoTermConnMonitorInitiatorInfoEv TA** Cause: CAUSE_SNAPSHOT address:B, device name: TB	GC1: Calling: X Called: A LRP: null Current calling: X Current called: A

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.getDafaultIntercomTarget DN()、CiscoIntercomAddress.getDefaultIntercomTarget Label()、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。	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	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。
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。	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	
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 getCurrentCallingPartyInfoNumberTyp e ().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():9180 28520261 getCurrentCalledPartyInfo(): 918028520261 getGlobalizedCallingParty (): 2222 getCurrentCallingPartyInfoNum berType().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 getCurrentCallingPartyInfoNumberTypе().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	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。
	GC3: ConsultCallActiveEv	
	GC3: ConnCreatedEv A	
	GC3: ConnCreatedEv D	
	GC3: CallCtlConnAlerting D	
	GC3: ConnConnectedEv D	
	GC3: CallCtlConnEstablishedEv B	
	CiscoConferenceStartEv GC3->GC1	
	GC3: CallCtlConnDisconnectedEv A	
	GC3: CallCtlConnDisconnectedEv D	
	GC1: ConnCreatedEv D	
	GC1: CallCtlConnEstablishedEv D	
	GC1: TermConnTalkingEv TermA	
	GC3: CallInvalidEv	
	CiscoConferenceEndEvent	
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。	GC1: ConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	Calling address: A Called address: B
	GC1: CallCtlConnDisconnectedEv D CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	
	GC1: ConnDisconnectedEv C CiscoFeatureReason = REASON_CONFERENCE Cause: CAUSE_NORMAL	

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。	**GC2-CallCtlConnDialingEv-C** **GC2-ConnCreatedEvent-PU** **GC2-ConnInprogressEvent-PU** **GC2-CallCtlConnEstablishedEv-C** GC2-CiscoCallChangedEv	CCalling: C, CCalled: NONE REASON_CALLPICKUP CCalling: C, CCalled: PU, LRP: PU CCalling C, CCalled: PU CCalling: A, CCalled: C, LRP: B Calling: A, Called: B REASON_CALLPICKUP
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。	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 2 から削除される。 C が Call 2 からドロップされる。	**GC2-ConnDisconnectedEvent-PU** **GC2-CallCtlConnDisconnectedEv-PU** GC2-TermConnDroppedEv GC2-CallCtlTermConnDroppedEv GC2-ConnDisconnectedEvent-C GC2-CallCtlConnDisconnectedEv-C GC2-CallInvalidEvent GC2-CallObservationEndedEv	CCalling: A, CCalled C, LRP: B REASON_CALLPICKUP CCalling: A, CCalled C, LRP: B 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。	GC1-ConnDisconnectedEvent-PU GC1-CallCtlConnDisconnectedEv-PU GC1-TermConnDroppedEv GC1-CallCtlTermConnDroppedEv GC1-ConnDisconnectedEvent-B GC1-CallCtlConnDisconnectedEv-B	

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
	GC1-ConnAlertingEvent GC1-CallCtlConnAlertingEv GC1-TermConnCreatedEvent	REASON_NORMAL
C が呼び出し中になる。		
	GC1-TermConnRingingEvent GC1-CallCtlTermConnRingingEv GC1-ConnConnectedEvent GC1-CallCtlConnEstablishedEv GC1-TermConnActiveEvent GC1-CallCtlTermConnTalkingEv	CCalling: A, CCalled: C, LRP: B REASON_NORMAL
C がピックアップする。		

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
	GC1-CallCtlConnEstablishedEv-A GC1-ConnCreatedEvent-B GC1-ConnInprogressEv-B GC1-CallCtlConnOfferedEv-B	CCalling A, CCalled B, Called: NOT SET LRP: NONE
B が呼び出し中になる。	GC1-ConnAlertingEvent-B GC1-CallCtlConnAlertingEv-B GC1-ConnCreatedEvent-C GC1-ConnInprogressEvent-C GC1-CallCtlConnOfferedEv-C	CCalling: A, CCalled: C, LRP: B Called: NOT SET REASON_CALLPICKUP
C が呼び出し中になる。	GC1-ConnAlertingEvent-C GC1-CallCtlConnAlertingEv-C	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。	GC1 CallActiveEv	calling: C
	GC1 ConnCreatedEv C:	lastRedirected:RP
ルート ポイント RP を登録する。	GC1 ConnConnectedEv C	called: A
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	GC1 CallCtlConnInitiatedEv C:	
	GC1 TermConnCreatedEv TC	
選択したルート：B、.....CSS：1、FP：　3	GC1 TermConnActiveEv TC	
選択したルート：D、.....CSS：1、FP：　1	GC1 CallCtlTermConnTalkingEv TC	
C が RP をコールする。	GC1 CallCtlConnDialingEv C:	
	GC1 CallCtlConnEstablishedEv C:	
A で呼び出し音が鳴る。	GC1 ConnCreatedEv RP:	
	GC1 ConnInProgressEv RP:	
A が応答する。C-A コールが接続される。	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	

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	
ユーザ B は、cancel キーを押す。	GC2: CiscoCallFeatureCancelledEv	CiscoCallFeatureCancelledEv.getConsultCall() は GC1 を返す。

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

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。	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。	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。	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。	
A の表示名は「abc」で、A' の表示名は「xyz」。		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	Cause = CAUSE_NORMAL
		CallCtlTermConnDroppedEv-TB	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	Cause = CAUSE_NORMAL
		TermConnDropEv-TB	
		CallCtlTermConnDroppedEv-TB	CiscoFeatureReason = REASON_CONFERENCE
		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。	

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。	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(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。 アプリケーションは A の接続で、Connection.disconnect() を呼び出す。	イベントなし A（abc）が会議から削除される。 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。 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。	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 GC2: CallActiveEv GC2: ConnCreatedEv **A:P1** GC2: ConnConnectedEv **A:P1** GC2: CallCtlConnInitiatedEv **A:P1**
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。	
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：	**端末オブザーバに配信されるイベント**
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。	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。	
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。	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：	**A の CallObserver に配信されるイベント**
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。	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、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。	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
アプリケーションが転送を実行する。	CiscoTransferStartEv getTransferControllerAddress() は A を返す。 ... CiscoTransferEndEv

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。

APPENDIX B

Cisco Unified JTAPI クラスおよびインターフェイス

この付録には、Cisco Unified JTAPI 実装において利用可能なすべてのクラスおよびインターフェイスの一覧があります。

- 「Cisco Unified JTAPI バージョン 1.2 クラスおよびインターフェイス」（P.B-1）には、JTAPI v 1.2 のすべてのクラスおよびメソッドの一覧が収録されています。サポートされるクラスおよびメソッドは、Cisco Unified JTAPI サポートのカラムに明記されています。
- 「Cisco Unified JTAPI 拡張クラスおよびインターフェイス」（P.B-21）には、Cisco Unified JTAPI 拡張クラスおよびメソッドの一覧が収録されています。
- 「Cisco トレース ログ クラスとインターフェイス」（P.B-24）には、エラー トレース クラスおよびメソッドの一覧が収録されています。

Cisco Unified JTAPI バージョン 1.2 クラスおよびインターフェイス

このセクションの内容は次のとおりです。

コア パッケージ

表 B-1 では、JTAPI コア パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-1　javax.telephony のサポート

クラス名	メソッド名	Cisco Unified JTAPI サポート	コメント
Address	addCallObserver	サポートする	
	addressObserver	サポートする	
	getAddressCapabilities	サポートする	
	getCallObservers	サポートする	
	getCapabilities	サポートする	
	getConnections	サポートする	
	getName	サポートする	
	getObservers	サポートする	
	getProvider	サポートする	
	getTerminals	サポートする	
	removeCallObserver	サポートする	
	removeObserver	サポートする	
AddressObserver	addressChangedEvent	サポートする	
Call	addObserver	サポートする	
	connect	サポートする	コールを発信する端末またはアドレス用に、CallObserver が存在している必要がある。 FeaturePriority パラメータはサポートされない。
	getCallCapabilities	サポートする	

表 B-1　javax.telephony のサポート （続き）

	getCapabilities	サポートする	
	getConnections	サポートする	
	getObservers	サポートする	
	getProvider	サポートする	
	getState	サポートする	
	removeObserver	サポートする	
CallObserver	callChangedEvent	サポートする	
Connection	disconnect	サポートする	
	getAddress	サポートする	
	getCall	サポートする	
	getCapabilities	サポートする	
	getConnectionCapabilities	サポートする	
	getState	サポートする	
	getTerminalConnections	サポートする	
JtapiPeer	getName	サポートする	
	getProvider	サポートする	
	getServices	サポートする	
JtapiPeerFactory	getJtapiPeer	サポートする	
Provider	addObserver	サポートする	
	createCall	サポートする	
	getAddress	サポートする	
	getAddressCapabilities()	サポートする	

表 B-1　javax.telephony のサポート （続き）

	getAddressCapabilities(Terminal)	サポートする	
	getAddresses	サポートする	
	getCallCapabilities()	サポートする	
	getCallCapabilities(Terminal, Address)	サポートする	
	getCalls	サポートする	このメソッドは、CallObservers がアドレスまたは端末に付加されているとき、ルーティングに RouteAddress が登録されているとき、または CiscoMediaTerminal が登録されているときにだけ、コールを返す。
	getCapabilities	サポートする	
	getConnectionCapabilities()	サポートする	
	getConnectionCapabilities(Terminal, Address)	サポートする	
	getName	サポートする	
	getObservers	サポートする	
	getProviderCapabilities()	サポートする	
	getProviderCapabilities(Terminal)	サポートする	
	getState	サポートする	
	getTerminal	サポートする	
	getTerminalCapabilities()	サポートする	
	getTerminalCapabilities(Terminal)	サポートする	
	getTerminalConnectionCapabilities()	サポートする	
	getTerminalConnectionCapabilities(Terminal)	サポートする	
	getTerminals	サポートする	

表 B-1　javax.telephony のサポート （続き）

	removeObserver	サポートする	
	shutdown	サポートする	
ProviderObserver	providerChangedEvent	サポートする	
Terminal	addCallObserver	サポートする	
	addObserver	サポートする	
	getAddresses	サポートする	
	getCallObservers	サポートする	
	getCapabilities	サポートする	
	getName	サポートする	
	getObservers	サポートする	
	getProvider	サポートする	
	getTerminalCapabilities	サポートする	
	getTerminalConnections	サポートする	
	removeCallObserver	サポートする	
	removeObserver	サポートする	
TerminalConnection	answer	サポートする	
	getCapabilities	サポートする	
	getConnection	サポートする	
	getState	サポートする	
	getTerminal	サポートする	
	getTerminalConnection Capabilities	サポートする	
TerminalObserver	terminalChangedEvent	サポートする	

コール センター パッケージ

表 B-2 では、JTAPI コール センター パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-2　javax.telephony.callcenter のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
ACDAddress	getACDManagerAddress		
	getLoggedOnAgents		
	getNumberQueued		
	getOldestCallQueued		
	getQueueWaitTime		
	getRelativeQueueLoad		
ACDAddressObserver			
ACDConnection	getACDManagerConnection		
ACDManagerAddress	getACDAddresses		
ACDManagerConnection	getACDConnections		
Agent	getACDAddress		
	getAgentAddress		
	getAgentID		
	getAgentTerminal		
	getState		
	setState		
AgentTerminal	addAgent		
	getAgents		
	removeAgents		
	setAgents		
AgentTerminalObserver			
CallCenterAddress	addCallObserver		
CallCenterCall	connectPredictive		
	getApplicationData		
	getTrunks		
	setApplicationData		
CallCenterCallObserver			
CallCenterProvider	getACDAddresses		
	getACDManagerAddresses		
	getRouteableAddresses		
CallCenterTrunk	getCall		

表 B-2　javax.telephony.callcenter のサポート （続き）

	getName		
	getState		
	getType		
RouteAddress	cancelRouteCallback	サポートする	
	getActiveRouteSessions	サポートする	
	getRouteCallback	サポートする	
	registerRouteCallback	サポートする	
RouteCallback	reRouteEvent	サポートする	
	routeCallbackEndedEvent	サポートする	
	routeEndEvent	サポートする	
	routeEvent	サポートする	
	routeUsedEvent	サポートする	
RouteSession	endRoute	サポートする	
	getCause	サポートする	
	getRouteAddress	サポートする	
	getState	サポートする	
	selectRoute	サポートする	

コール センター機能パッケージ

表 B-3 では、JTAPI コール センター機能パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-3　javax.telephony.callcenter.capabilities のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
ACDAddressCapabilities	canGetACDManagerAddress		

表 B-3　javax.telephony.callcenter.capabilities のサポート （続き）

	canGetLoggedOnAgents		
	canGetNumberQueued		
	canGetOldestCallQueued		
	canGetQueueWaitTime		
	canGetRelativeQueueLoad		
ACDConnectionCapabilities	canGetACDManager Connection		
ACDManagerAddress Capabilities	canGetACDAddresses		
ACDManagerConnection Capabilities	canGetACDConnections		
AgentTerminalCapabilities	canHandleAgents		
CallCenterAddress Capabilities	canAddCallObserver		
CallCenterCallCapabilities	canConnectPredictive		
	canGetTrunks		
	canHandleApplicationData		
CallCenterProvider Capabilities	canGetACDAddresses	サポートする	
	canGetACDManagerAddresses	サポートする	
	canGetRouteableAddresses	サポートする	
RouteAddressCapabilities	canRouteCalls	サポートする	

コール センター イベント パッケージ

表 B-4 では、JTAPI コール センター イベント パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-4　javax.telephony.callcenter.events のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
ACDAddrBusyEv			
ACDAddrEv	getAgent		
	getAgentAddress		
	getAgentTerminal		
	getState		

表 B-4　javax.telephony.callcenter.events のサポート （続き）

	getTrunks		
ACDAddrLoggedOffEv			
ACDAddrLoggedOnEv			
ACDAddrNotReadyEv			
ACDAddrReadyEv			
ACDAddrUnknownEv			
ACDAddrWorkNotReadyEv			
ACDAddrWorkReadyEv			
AgentTermBusyEv			
AgentTermEv	getACDAddress		
	getAgent		
	getAgentAddress		
	getAgentID		
	getState		
AgentTermLoggedOffEv			
AgentTermLoggedOnEv			
AgentTermNotReadyEv			
AgentTermReadyEv			
AgentTermUnknownEv			
AgentTermWorkNotReadyEv			
AgentTermWorkReadyEv			
CallCentCallAppDataEv	getApplicationData		
CallCentCallEv	getCalledAddress		
	getCallingAddress		
	getCallingTerminal		
	getLastRedirectedAddress		
	getTrunks		
CallCentConnEv			
CallCentConnInProgressEv			
CallCentEv	getCallCenterCause		
CallCentTrunkEv	getTrunk		
CallCentTrunkInvalidEv			
CallCentTrunkValidEv			
ReRouteEvent		サポートする	
RouteCallbackEndedEvent	getRouteAddress	サポートする	
RouteEndEvent		サポートする	

表 B-4　javax.telephony.callcenter.events のサポート （続き）

RouteEvent	getCallingAddress	サポートする	
	getCallingTerminal	サポートする	
	getCurrentRouteAddress	サポートする	
	getRouteSelectAlgorithm	サポートする	
	getSetupInformation	サポートする	
RouteSessionEvent	getRouteSession	サポートする	
RouteUsedEvent	getCallingAddress	サポートする	
	getCallingTerminal	サポートする	
	getDomain	サポートする	
	getRouteUsed	サポートする	

コール制御パッケージ

表 B-5 では、JTAPI コール制御パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-5　javax.telephony.callcontrol のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
CallControlAddress	cancelForwarding	サポートする	Call Forward All の場合だけ
	getDoNotDisturb		
	getForwarding	サポートする	Call Forward All の場合だけ
	getMessageWaiting		
	setDoNotDisturb		
	setForwarding	サポートする	Call Forward All の場合だけ
	setMessageWaiting		
CallControlCall	addParty		

表 B-5　javax.telephony.callcontrol のサポート （続き）

	conference	サポートする	コンサルト会議のシナリオでは、OriginalCall.conference (ConsultCall) だけがサポートされる。ConsultCall.conference (OriginalCall) はサポートされない。
	consult(TerminalConnection)	サポートする	
	consult(TerminalConnection, String)	サポートする	
	drop	サポートする	
	getCalledAddress	サポートする	
	getCallingAddress	サポートする	
	getCallingTerminal	サポートする	
	getConferenceController	サポートする	
	getConferenceEnable	サポートする	
	getLastRedirectedAddress	サポートする	
	getTransferController	サポートする	
	getTransferEnable	サポートする	
	offHook	サポートする	
	setConferenceController	サポートする	
	setConferenceEnable	サポートする	
	setTransferController	サポートする	
	setTransferEnable	サポートする	

表 B-5　javax.telephony.callcontrol のサポート （続き）

	transfer(Call)	サポートする	コンサルト転送のシナリオでは、OriginalCall.transfer (ConsultCall) だけがサポートされる。ConsultCall.transfer (OriginalCall) はサポートされない。
	transfer(String)	サポートする	
CallControlCallObserver		サポートする	
CallControlConnection	accept	サポートする	
	addToAddress	サポートする	
	getCallControlState	サポートする	
	park	サポートする	
	リダイレクト	サポートする	CallControlConnection.ESTABLISHED 状態にある接続をリダイレクトできる。
	reject	サポートする	
CallControlForwarding	getDestinationAddress		
	getFilter		
	getSpecificCaller		
	getType		
CallControlTerminal	getDoNotDisturb		
	pickup (Address, Address)		
	pickup (Connection, Address)		
	pickup (TerminalConnection, Address)		
	pickupFromGroup(Address)		
	pickupFromGroup(String, Address)		
	setDoNotDisturb		
CallControlTerminalConnection	getCallControlState	サポートする	
	hold	サポートする	

表 B-5　javax.telephony.callcontrol のサポート （続き）

	join	サポートする	CiscoIntercomAddresses に対してだけ実装。
	leave		
	unhold	サポートする	
CallControlTerminalObserver			

コール制御機能パッケージ

表 B-6 では、JTAPI コール制御機能パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-6　javax.telephony.callcontrol.capabilities のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
CallControlAddress Capabilities	canCancelForwarding	サポートする	
	canGetDoNotDisturb	サポートする	
	canGetForwarding	サポートする	
	canGetMessageWaiting	サポートする	
	canSetDoNotDisturb	サポートする	
	canSetForwarding	サポートする	
	canSetMessageWaiting	サポートする	
CallControlCallCapabilities	canAddParty	サポートする	
	canConference	サポートする	
	canConsult	サポートする	
	canConsult(TerminalConnection)	サポートする	
	canConsult(TerminalConnection, String)	サポートする	

表 B-6　javax.telephony.callcontrol.capabilities のサポート （続き）

	canDrop	サポートする	
	canOffHook	サポートする	
	canSetConferenceController	サポートする	
	canSetConferenceEnable	サポートする	
	canSetTransferController	サポートする	
	canSetTransferEnable	サポートする	
	canTransfer	サポートする	
	canTransfer(Call)	サポートする	
	canTransfer(String)	サポートする	
CallControlConnection Capabilities	canAccept	サポートする	
	canAddToAddress	サポートする	
	canPark	サポートする	
	canRedirect	サポートする	
	canReject	サポートする	
CallControlTerminal Capabilities	canGetDoNotDisturb	サポートする	
	canPickup	サポートする	
	canPickup(Address, Address)	サポートする	
	canPickup(Connection, Address)	サポートする	
	canPickup(TerminalConnection, Address)	サポートする	
	canPickupFromGroup	サポートする	
	canPickupFromGroup(Address)	サポートする	
	canPickupFromGroup(String, Address)	サポートする	

表 B-6　javax.telephony.callcontrol.capabilities のサポート （続き）

	canSetDoNotDisturb	サポートする	
CallControlTerminal ConnectionCapabilities	canHold	サポートする	
	canJoin	サポートする	
	canLeave	サポートする	
	canUnhold	サポートする	

コール制御イベント パッケージ

表 B-7 では、JTAPI コール制御イベント パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-7　javax.telephony.callcontrol.events のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
CallCtlAddrDoNotDisturbEv	getDoNotDisturbState		
CallCtlAddrEv			
CallCtlAddrForwardEv	getForwarding	サポートする	
CallCtlAddrMessageWaitingEv	getMessageWaitingState		
CallCtlCallEv	getCalledState	サポートする	
	getCallingAddress	サポートする	
	getCallingTerminal	サポートする	
	getLastRedirectedAddress	サポートする	
CallCtlConnAlertingEv		サポートする	
CallCtlConnDialingEv	getDigits	サポートする	
CallCtlConnDisconnectedEv		サポートする	
CallCtlConnEstablishedEv		サポートする	

表 B-7　javax.telephony.callcontrol.events のサポート （続き）

CallCtlConnEv		サポートする	
CallCtlConnFailedEv		サポートする	
CallCtlConnInitiatedEv		サポートする	
CallCtlConnNetworkAlertingEv		サポートする	
CallCtlConnNetworkReachedEv		サポートする	
CallCtlConnOfferedEv		サポートする	
CallCtlConnQueuedEv	getNumberInQueue	サポートする	
CallCtlConnUnknownEv		サポートする	
CallCtlEv	getCallControlCause	サポートする	
CallCtlTermConnBridgedEv			
CallCtlTermConnDroppedEv		サポートする	
CallCtlTermConnEv		サポートする	
CallCtlTermConnHeldEv		サポートする	
CallCtlTermConnInUseEv			
CallCtlTermConnRingingEv		サポートする	
CallCtlTermConnTalkingEv		サポートする	
CallCtlTermConnUnknownEv		サポートする	
CallCtlTermDoNotDisturbEv			
CallCtlTermEv			

機能パッケージ

表 B-8 では、JTAPI 機能パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-8　javax.telephony.capabilities のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
AddressCapabilities	isObservable	サポートする	
CallCapabilities	canConnect	サポートする	
	isObservable	サポートする	
ConnectionCapabilities	canDisconnect	サポートする	
ProviderCapabilities	isObservable	サポートする	
TerminalCapabilities	isObservable	サポートする	
TerminalConnection Capabilities	canAnswer	サポートする	

イベント パッケージ

表 B-9 では、JTAPI イベント パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-9　javax.telephony.events のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
AddrEv	getAddress	サポートする	
AddrObservationEndedEv		サポートする	
CallActiveEv		サポートする	
CallEv	getCall	サポートする	
CallInvalidEv		サポートする	
CallObservationEndedEv	getEndedObject	サポートする	
ConnAlertingEv		サポートする	

表 B-9　javax.telephony.events のサポート （続き）

ConnConnectedEv		サポートする	
ConnCreatedEv		サポートする	
ConnDisconnectedEv		サポートする	
ConnEv	getConnection	サポートする	
ConnFailedEv		サポートする	
ConnInProgressEv		サポートする	
ConnUnknownEv		サポートする	
Ev	getCause	サポートする	
	getID	サポートする	
	getMetaCode	サポートする	
	getObserved	サポートする	
	isNewMetaEvent	サポートする	
ProvEv	getProvider	サポートする	
ProvInServiceEv		サポートする	
ProvObservationEndedEv		サポートする	
ProvOutOfServiceEv		サポートする	
ProvShutdownEv		サポートする	
TermConnActiveEv		サポートする	
TermConnCreatedEv		サポートする	
TermConnDroppedEv		サポートする	
TermConnEvgetTerminal Connection		サポートする	
TermConnPassiveEv			

表 B-9　javax.telephony.events のサポート （続き）

TermConnRingingEv		サポートする	
TermConnUnknownEv		サポートする	
TermEv	getTerminal	サポートする	
TermObservationEndedEv		サポートする	

メディア パッケージ

表 B-10 では、JTAPI メディア パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-10　javax.telephony.media のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
MediaCallObserver		サポートする	
MediaTerminalConnection	generateDtmf	サポートする	
	getMediaAvailability		
	getMediaState		
	setDtmfDetection	サポートする	
	startPlaying		
	startRecording		
	stopPlaying		
	stopRecording		
	useDefaultMicrophone		
	useDefaultSpeaker		
	usePlayURL		
	useRecordURL		

メディア機能パッケージ

表 B-11 では、JTAPI メディア機能パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-11　javax.telephony.media.capabilities のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
MediaTerminalConnection Capabilities	canDetectDtmf	サポートする	
	canGenerateDtmf	サポートする	
	canStartPlaying	サポートする	
	canStartRecording	サポートする	
	canStopPlaying	サポートする	
	canStopRecording	サポートする	
	canUseDefaultMicrophone	サポートする	
	canUseDefaultSpeaker	サポートする	
	canUsePlayURL	サポートする	
	canUseRecordURL	サポートする	

メディア イベント パッケージ

表 B-12 では、JTAPI メディア イベント パッケージの各 JTAPI インターフェイスとそれに続いて関連するメソッドをリストし、そのクラスが Cisco Unified JTAPI 実装でサポートされているかどうかを示しています。

表 B-12　javax.telephony.media.events のサポート

クラス名	メソッド名	Cisco Unified JTAPI のサポート	コメント
MediaEv	getMediaCause	サポートする	
MediaTermConnAvailableEv			
MediaTermConnDtmfEv	getDtmfDigit	サポートする	
MediaTermConnEv		サポートする	

表 B-12　javax.telephony.media.events のサポート （続き）

MediaTermConnStateEv	getMediaState		
MediaTermConnUnavailableEv			

サポートされないパッケージ

表 B-13 は、Cisco Unified JTAPI 実装でサポートされない JTAPI パッケージを示しています。

表 B-13　サポートされない JTAPI パッケージ

サポートされない JTAPI パッケージ
JTAPI 電話パッケージ
JTAPI 電話機能パッケージ
JTAPI 電話イベント パッケージ
JTAPI プライベート データ パッケージ
JTAPI プライベート データ機能パッケージ
JTAPI プライベート データ イベント パッケージ

Cisco Unified JTAPI 拡張クラスおよびインターフェイス

Cisco Unified JTAPI 拡張クラス

表 B-14　Cisco Unified JTAPI 拡張クラス

Cisco 拡張クラス	メソッド名
CiscoMediaCapability	getMaxFramesPerPacket() getPayloadType() toString()
CiscoG711MediaCapability	
CiscoG723MediaCapability	getBitRate() toString()
CiscoGSMMediaCapability	
RegistrationException	
UnregistrationException	

Cisco Unified JTAPI 拡張インターフェイス

表 B-15　Cisco Unified JTAPI 拡張インターフェイスおよびそのメソッド

Cisco 拡張インターフェイス	メソッド名
CiscoAddrCreatedEv	getAddress()
CiscoAddress	getType()
CiscoAddressObserver	
CiscoAddrEv	
CiscoAddrInService	
CiscoAddrOutOfService	
CiscoCall	getCallID()
CiscoCallEv	
CiscoCallID	getCall() intValue()
CiscoConferenceEndEv	getConferenceCall() getFinalCall() getHeldConferenceController() getTalkingConferenceController()
CiscoConferenceStartEv	getConferenceCall() getFinalCall() getHeldConferenceController() getTalkingConferenceController()
CiscoConnection	getConnectionID() getReason()
CiscoConnectionID	getConnection() intValue()
CiscoConsultCall	getConsultingTerminalConnection()
CiscoConsultCallActiveEv	getHeldTerminalConnection()
CiscoEv	
CiscoJtapiPeer	
CiscoMediaTerminal	getRTPInputProperties() getRTPOutputProperties() register(InetAddress, int) unregister()
CiscoProvEv	
CiscoProvider	getCallbackGuardEnabled() getMediaTerminal() getMediaTerminals() setCallbackGuardEnabled()

表 B-15　Cisco Unified JTAPI 拡張インターフェイスおよびそのメソッド （続き）

Cisco 拡張インターフェイス	メソッド名
CiscoProviderObserver	
CiscoRouteSession	getCall()
CiscoRTPInputProperties	getBitRate() getEchoCancellation() getLocalAddress() getLocalPort() getPacketSize() getPayloadType()
CiscoRTPInputStartedEv	getRTPInputProperties()
CiscoRTPInputStoppedEv	
CiscoRTPOutputProperties	getBitRate() getMaxFramesPerPacket() getPacketSize() getPayloadType() getPrecedenceValue() getRemoteAddress() getRemotePort()
CiscoRTPOutputStartedEv	getRTPOutputProperties()
CiscoRTPOutputStoppedEv	
CiscoSynronousObserver	
CiscoTermCreatedEv	getTerminal()
CiscoTermEv	
CiscoTerminal	getRegistrationState()
CiscoTerminalConnection	
CiscoTerminalObserver	
CiscoTermInServiceEv	
CiscoTermOutOfServiceEv	
CiscoTransferEndEv	getFinalCall() getTransferController() getTransferredCall()
CiscoTransferStartEv	getFinalCall() getTransferController() getTransferredCall()
ObjectContainer	getObject() setObject()
RTPBitRate	
RTPPayload	

Cisco トレース ログ クラスとインターフェイス

Cisco トレース ログ クラス

表 B-16　Cisco トレース ログ クラス

Cisco トレース ログ クラス	メソッド名
LogFileOutputStream	close()
	flush()
	getCurrentFile()
	getFileExtension()
	getFileNameBase()
	getMaxFiles()
	getMaxFileSize()
	write(byte[], int, int)
	write(int)
NullTraceWriter	close()
	flush()
	getEnabled()
	print(String)
	println(String)
OutputStreamTraceWriter	close()
	flush()
	getEnabled()
	print(String)
	println(String)
	setOutputStream(OUputStream
TraceManagerFactory	getModules()
	registerModule(String)
	registerModule(TraceModule)
	registerModule(TraceModule, OutputStream)

Cisco トレース ログ インターフェイス

表 B-17　Cisco トレース ログ インターフェイス

Cisco トレース ログ インターフェイス	メソッド名
ConditionalTrace	disable()
	enable()
Trace	append(Object)
	append(String)
	getName()
	isEnabled()
	print(Object)
	print(String)
	print(String, Object)
	print(String, String)
	println(Object)
	println(String)
	println(String, Object)
	println(String, String)
	setDefaultMnemonic(String)

表 B-17　Cisco トレース ログ インターフェイス （続き）

Cisco トレース ログ インターフェイス	メソッド名
TraceManager	disableAll()
	disableTimeStamp()
	enableAll()
	enableTimeStamp()
	getConditionalTrace(String)
	getConditionalTrace(String, String)
	getName()
	getOutputStream()
	getSubFacilities()
	getTraces()
	getTraceWriter()
	getUnconditionalTrace(String)
	getUnconditionalTrace(String, String)
	removeTrace(String)
	removeTrace(Trace)
	setOutputStream(OutputStream)
	setSubFacilities()
	setTraceWriter()
TraceModule	getTraceManager()
	getTraceModuleName()
TRACETYPE	
TraceWriter	close()
	flush()
	getEnabled()
	print(String)
	println(String)
UnconditionalTrace	

APPENDIX C

Cisco Unified JTAPI のトラブルシューティング

この付録には、CTI エラー コードや、CiscoEvent ID など、トラブルシューティングに役立つ情報が記載されています。内容は、次のとおりです。

CTI エラー コード

エラー コード：	説明
ASSOCIATED_LINE_NOT_OPEN	このエラーは、オープンされていない回線上で要求が発行されたことを示します。
CALL_ALREADY_EXISTS	このエラーは、回線上にすでに別のコールが存在していることを示します。
CALL_DROPPED	機能（保留、保留解除、転送、または会議）要求の後、要求が完了する前にコールがドロップされました。
CALLHANDLE_NOTINCOMINGCALL	このエラーは、存在しないか、または正しい状態にないコールに応答しようとしたことを示します。
CALLHANDLE_UNKNOWN_TO_LINECONTROL	このエラーは、回線制御側で検知されていないコールをリダイレクトしようとしたことを示します。
CANNOT_OPEN_DEVICE	このエラーは、関連するデバイスが登録解除のため、デバイスのオープンに失敗したことを示します。
CANNOT_TERMINATE_MEDIA_ON_PHONE	このエラーは、デバイスに実際の電話がある場合に、メディアを終端できない（常に、電話がメディアを終端する）ことを示します。
CFWDALL_ALREADY_SET	このエラーは、すでにオンになっている CFWALL をオンにしようとしたことを示します。

エラー コード：	説明
CFWDALL_DESTN_INVALID	このエラーは、無効な宛先に CFWALL を実行しようとしたことを示します。
CLUSTER_LINK_FAILURE	このエラーは、クラスタ内にある Cisco Unified CM の 1 つへのリンクが失敗したことを示します（ネットワーク エラー）。
COMMAND_NOT_IMPLEMENTED_ON_DEVICE	このエラーは、デバイスがコマンドをサポートしていないことを示します。
CONFERENCE_ALREADY_PRESENT	このエラーは、すでに会議に参加している通話者に会議を実行しようとしたことを示します。
CONFERENCE_FAILED	このエラーは、会議の開催が成功しなかったことを示します。
CONFERENCE_FULL	このエラーは、すべての会議ブリッジが使用中であることを示します。
CONFERENCE_INACTIVE	このエラーは、コンサルト会議が有効ではないときに、会議を開催しようとしたことを示します。
CONFERENCE_INVALID_PARTICIPANT	このエラーは、自分自身または無効な関係者に会議を実行しようとしたことを示します。
CTIERR_ACCESS_TO_DEVICE_DENIED	このエラーは、デバイスへのアクセスが拒否されたことを示します。
CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	このエラーは、アプリケーションのソフトキーが別のアプリケーションによってすでに制御されていることを示します。
CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	このエラーは、アプリケーションのデータ サイズが限度を超えていることを示します。
CTIERR_BIB_NOT_CONFIGURED	このエラーは、ビルトインブリッジ（BIB）が構成されていないことを示します。
CTIERR_BIB_RESOURCE_NOT_AVAILABLE	このエラーは、ビルトインブリッジ（BIB）リソースが利用できないことを示します。
CTIERR_CALL_MANAGER_NOT_AVAILABLE	このエラーは、Communications Manager が現在利用できないことを示します。
CTIERR_CALL_NOT_EXISTED	このエラーは、コールが存在していないことを示します。
CTIERR_CALL_PARK_NO_DN	このエラーは、コール パーク DN がないことを示します。
CTIERR_CALL_REQUEST_ALREADY_OUTSTANDING	このエラーは、コール要求が未処理であることを示します。
CTIERR_CALL_UNPARK_FAILED	このエラーは、コールのパーク解除が失敗したことを示します。
CTIERR_CAPABILITIES_DO_NOT_MATCH	このエラーは、機能が適合しないことを示します。
CTIERR_CLOSE_DELAY_NOT_SUPPORTED_WITH_REG_TYPE	このエラーは、この登録タイプではクローズ遅延がサポートされないことを示します。
CTIERR_CONFERENCE_ALREADY_EXISTED	このエラーは、会議がすでに存在していることを示します。
CTIERR_CONFERENCE_NOT_EXISTED	このエラーは、会議が存在していないことを示します。

エラー コード：	説明
CTIERR_CONNECTION_ON_INVALID_PORT	このエラーは、アプリケーションが無効なポートに接続しようとしていることを示します。
CTIERR_CONSULT_CALL_FAILURE	このエラーは、コンサルト コールが失敗したことを示します。
CTIERR_CONSULTCALL_ALREADY_OUTSTANDING	このエラーは、コンサルト コールが未処理であることを示します。
CTIERR_CRYPTO_CAPABILITY_MISMATCH	このエラーは、デバイスの暗号アルゴリズムが現在のデバイス登録と適合してないためにデバイス登録が失敗したことを示します。
CTIERR_CTIHANDLER_PROCESS_CREATION_FAILED	このエラーは、CTIHandler プロセスの作成が失敗したことを示します。
CTIERR_DB_INITIALIZATION_ERROR	このエラーは、DB 初期化エラーを示します。
CTIERR_DEVICE_ALREADY_OPENED	このエラーは、デバイスがすでにオープンされていることを示します。
CTIERR_DEVICE_NOT_OPENED_YET	このエラーは、デバイスがまだオープンされていないことを示します。
CTIERR_DEVICE_OWNER_ALIVE_TIMER_STARTED	このエラーは、デバイスの登録に失敗したことを示します。
CTIERR_DEVICE_REGISTRATION_FAILED_NOT_SUPPORTED_MEDIATYPE	このエラーは、無効なメディア タイプであることを示します。インターコム回線の場合、CTIPort はダイナミック メディア ポート登録で登録する必要があります。
CTIERR_DEVICE_RESTRICTED	このエラーは、デバイスが制限されていることを示します。
CTIERR_DEVICE_SHUTTING_DOWN	このエラーは、デバイスがシャット ダウン中であることを示します。
CTIERR_DIRECTORY_LOGIN_TIMEOUT	このエラーは、ディレクトリのログインがタイム アウトになっていることを示します。
CTIERR_DUPLICATE_CALL_REFERENCE	このエラーは、コールの参照値が重複していることを示します。
CTIERR_DYNREG_IPADDRMODE_MISMATCH	これは、シスコのメディア/ルート端末が複数のアドレッシング モードで登録されている場合に登録に失敗したことを示します。
CTIERR_FAC_CMC_REASON_CMC_INVALID	入力した Client Matter Code (CMC) が無効です。
CTIERR_FAC_CMC_REASON_CMC_NEEDED	コールをオファーするのに CMC が必要です。
CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	コールをオファーするのに Forced Authorization Code（FAC）および CMC が必要です。
CTIERR_FAC_CMC_REASON_FAC_INVALID	入力された FAC が無効です。
CTIERR_FAC_CMC_REASON_FAC_NEEDED	コールをオファーするのに FAC が必要です。
CTIERR_FEATURE_ALREADY_REGISTERED	このエラーは、機能がすでに登録されていることを示します。
CTIERR_FEATURE_DATA_REJECT	このエラーは、機能データが拒否されたことを示します。
CTIERR_FEATURE_SELECT_FAILED	このエラーは、機能の選択に失敗したことを示します。

エラー コード：	説明
CTIERR_ILLEGAL_DEVICE_TYPE	このエラーは、デバイス タイプが正しくないことを示します。
CTIERR_INCOMPATIBLE_AUTOINSTALL_PROTOCOL_VERSION	このエラーは、自動インストール プロトコルのバージョンに互換性がないことを示します。
CTIERR_INCORRECT_MEDIA_CAPABILITY	不正なメディア機能が原因で、デバイス登録が失敗しました。
CTIERR_INFORMATION_NOT_AVAILABLE	このエラーは、情報がないことを示します。
CTIERR_INTERCOM_SPEEDDIAL_ALREADY_CONFIGURED	このエラーは、インターコム ターゲットの値がすでに構成され、アプリケーションがインターコム ターゲット DN でコールを開始しようとしていることを示します。
CTIERR_INTERCOM_SPEEDDIAL_ALREADY_SET	このエラーは、インターコム ターゲットの値がすでに設定されており、アプリケーションが設定し直そうとしているためにインターコム要求が失敗したことを示します。
CTIERR_INTERCOM_SPEEDDIAL_DESTN_INVALID	このエラーは、インターコム ターゲットの値がインターコム グループ内にないためにインターコム要求が失敗したことを示します。
CTIERR_INTERCOM_TALKBACK_ALREADY_PENDING	このエラーは、インターコム応答要求がすでに処理中であることを示します。
CTIERR_INTERCOM_TALKBACK_FAILURE	このエラーは、何らかの理由で応答要求が失敗したことを示します。
CTIERR_INTERNAL_FAILURE	このエラーは、CTI 内部エラーが発生したことを示します。
CTIERR_INVALID_CALLID	このエラーは、コール ID が無効であることを示します。
CTIERR_INVALID_DEVICE_NAME	このエラーは、デバイス名が無効であることを示します。
CTIERR_INVALID_DTMFDIGITS	Play DTMF 要求が無効な番号であるために失敗しました。
CTIERR_INVALID_FILTER_SIZE	このエラーは、フィルタ サイズが無効であることを示します。
CTIERR_INVALID_MEDIA_DEVICE	このエラーは、メディア デバイスが無効であることを示します。
CTIERR_INVALID_MEDIA_PARAMETER	このエラーは、メディア パラメータが無効であることを示します。
CTIERR_INVALID_MEDIA_PROCESS	このエラーは、無効なメディア プロセスが存在していることを示します。
CTIERR_INVALID_MEDIA_RESOURCE_ID	このエラーは、メディア リソース ID が無効であることを示します。
CTIERR_INVALID_MESSAGE_HEADER_INFO	このエラーは、ヘッダー情報が無効であることを示します。
CTIERR_INVALID_MESSAGE_LENGTH	このエラーは、メッセージの長さが無効であることを示します。

エラー コード：	説明
CTIERR_INVALID_MONITOR_DESTN	このエラーは、無効なモニタリング対象のためにモニタリング要求が失敗したことを示します。
CTIERR_INVALID_MONITOR_DN_TYPE	このエラーは、モニタリング DN タイプが無効であることを示します。
CTIERR_INVALID_MONITORMODE	このエラーは、モニタリングモードが無効であるためにモニタリング要求が失敗したことを示します。
CTIERR_INVALID_PARAMETER	このエラーは、パラメータが無効であることを示します。
CTIERR_INVALID_PARK_DN	このエラーは、DN が 無効なパーク DN であることを示します。
CTIERR_INVALID_PARK_REGISTRATION_HANDLE	このエラーは、ハンドルが無効なパーク登録ハンドルであることを示します。
CTIERR_INVALID_RESOURCE_TYPE	このエラーは、リソース タイプが無効であることを示します。
CTIERR_IPADDRMODE_MISMATCH	これは、IP アドレッシング モードが一致しないために登録に失敗したことを示します。
CTIERR_LINE_OUT_OF_SERVICE	このエラーは、回線がアウト オブ サービスであることを示します。
CTIERR_LINE_RESTRICTED	このエラーは、回線が制限されていることを示します。
CTIERR_MAXCALL_LIMIT_REACHED	このエラーは、最大コール制限値に達したことを示します。
CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC	このエラーは、デバイスを動的メディア終端で登録しようとしたためにデバイスの登録が失敗したことを示します。
CTIERR_MEDIA_ALREADY_TERMINATED_NONE	このエラーは、デバイスがすでにメディア終端がなしで登録されているためにデバイスの登録が失敗したことを示します。
CTIERR_MEDIA_ALREADY_TERMINATED_STATIC	このエラーは、デバイスを静的メディア終端で登録しようとしたためにデバイスの登録が失敗したことを示します。
CTIERR_MEDIA_CAPABILITY_MISMATCH	このエラーは、デバイスのメディア機能が現在のデバイス登録と適合してないためにデバイス登録が失敗ことを示します。
CTIERR_MEDIA_RESOURCE_NAME_SIZE_EXCEEDED	このエラーは、メディア リソース名のサイズが限度を超えていることを示します。
CTIERR_MEDIAREGISTRATIONTYPE_DO_NOT_MATCH	このエラーは、メディア登録のタイプが一致していないことを示します。
CTIERR_MESSAGE_TOO_BIG	このエラーは、メッセージが長すぎることを示します。
CTIERR_MORE_ACTIVE_CALLS_THAN_RESERVED	このエラーは、アクティブなコールが予約されたコールよりも多いことを示します。
CTIERR_NO_EXISTING_CALLS	このエラーは、コールが存在していないことを示します。
CTIERR_NO_EXISTING_CONFERENCE	このエラーは、会議が存在していないことを示します。
CTIERR_NO_RECORDING_SESSION	このエラーは、録音セッションがないために録音要求に失敗したことを示します。

エラー コード：	説明
CTIERR_NO_RESPONSE_FROM_MP	このエラーは、メディア リソースからの応答がないことを示します。
CTIERR_NOT_PRESERVED_CALL	このエラーは、コールが維持されていないことを示します。
CTIERR_OPERATION_FAILED_QUIETCLEAR	このエラーは、一時的な障害が原因で、このコールの機能が利用できないことを示します。
CTIERR_OPERATION_NOT_ALLOWED	このエラーは、この操作が許可されていないことを示します。
CTIERR_OUT_OF_BANDWIDTH	このエラーは、帯域幅の範囲外であることを示します。
CTIERR_OWNER_NOT_ALIVE	このエラーは、デバイスの登録が失敗したことを示します。
CTIERR_PENDING_ACCEPT_OR_ANSWER_REQUEST	このエラーは、保留中の受け入れ要求または応答要求があることを示します。
CTIERR_PENDING_START_MONITORING_REQUEST	このエラーは、保留中のモニタリング開始要求があることを示します。
CTIERR_PENDING_START_RECORDING_REQUEST	このエラーは、保留中の録音開始要求があることを示します。
CTIERR_PENDING_STOP_RECORDING_REQUEST	このエラーは、保留中の録音停止要求があることを示します。ことを示します。
CTIERR_PRIMARY_CALL_INVALID	このエラーは、モニタリング要求のプライマリ コールが無効であるか、またはアイドル状態であることを示します。
CTIERR_PRIMARY_CALL_STATE_INVALID	このエラーは、モニタリング要求のプライマリ コールが無効な状態であることを示します。
CTIERR_RECORDING_ALREADY_INPROGRESS	このエラーは、録音がすでに進行中であるために録音要求が失敗したことを示します。
CTIERR_RECORDING_CONFIG_NOT_MATCHING	このエラーは、録音設定が一致しないことを示します。
CTIERR_RECORDING_SESSION_INACTIVE	このエラーは、録音セッションが非アクティブであるために録音要求に失敗したことを示します。
CTIERR_REDIRECT_UNAUTHORIZED_COMMAND_USAGE	このエラーは、未承認のコマンド使用のリダイレクトを示します。
CTIERR_REGISTER_FEATURE_ACTIVATION_FAILED	このエラーは、登録機能のアクティベーションに失敗したことを示します。
CTIERR_REGISTER_FEATURE_APP_ALREADY_ REGISTERED	機能登録アプリケーションがすでに登録されています。
CTIERR_REGISTER_FEATURE_PROVIDER_NOT_ REGISTERED	機能登録プロバイダーが登録されていません。
CTIERR_RESOURCE_NOT_AVAILABLE	このエラーは、要求を処理するためにリソースを利用できないことを示します。
CTIERR_START_MONITORING_FAILED	このエラーは、モニタリング開始要求に失敗したことを示します。
CTIERR_START_RECORDING_FAILED	このエラーは、録音開始要求に失敗したことを示します。

エラー コード：	説明
CTIERR_STATION_SHUT_DOWN	このエラーは、ステーションのシャットダウンが存在することを示します。
CTIERR_SYSTEM_ERROR	このエラーは、CTI システム エラーを示します。
CTIERR_UDP_PASS_THROUGH_NOT_SUPPORTED	このエラーは、UDP データ パススルーがサポートされないことを示します。
CTIERR_UNKNOWN_EXCEPTION	このエラーは、不明な例外が発生したことを示します。
CTIERR_UNSUPPORTED_CALL_PARK_TYPE	このエラーは、コール パークのタイプがサポートされないことを示します。
CTIERR_UNSUPPORTED_CFWD_TYPE	このエラーは、コール転送タイプがサポートされないことを示します。
CTIERR_USER_NOT_AUTH_FOR_SECURITY	このエラーは、ユーザがセキュア接続のために認証されていないことを示します。
DARES_INVALID_REQ_TYPE	このエラーは、コール処理内部エラー、DaRes 無効要求タイプが存在することを示します。
DATA_SIZE_LIMIT_EXCEEDED	このエラーは、XML データ オブジェクト サイズが、許容値よりも大きいことを示します。
DB_ERROR	このエラーは、デバイス クエリーに不正なデバイス タイプがあることを示します。
DB_ILLEGAL_DEVICE_TYPE	このエラーは、DB に不正なデバイス タイプがあることを示します。
DB_NO_MORE_DEVICES	このエラーは使用されなくなりました。
DESTINATION_BUSY	このエラーは、転送先が通話中であることを示します。
DESTINATION_UNKNOWN	このエラーは、転送先が見つからないことを示します。
DEVICE_ALREADY_REGISTERED	このエラーは、デバイスがすでに登録されているためにデバイス登録が失敗したことを示します。
DEVICE_NOT_OPEN	このエラーは、デバイスがオープンされていないか、または登録されていないため、回線のオープンに失敗したことを示します。
DEVICE_OUT_OF_SERVICE	このエラーは、デバイスがアウト オブ サービスであることを示します。
DIGIT_GENERATION_ALREADY_IN_PROGRESS	このエラーは、番号の生成が進行中であることを示します。
DIGIT_GENERATION_CALLSTATE_CHANGED	このエラーは、コールの状態が無効で継続できないことを示します。
DIGIT_GENERATION_WRONG_CALL_HANDLE	このエラーは、コール ハンドルが無効で、コールが存在しない可能性があることを示します。
DIGIT_GENERATION_WRONG_CALL_STATE	このエラーは、コールの状態が無効で、番号を生成できないことを示します。
DIRECTORY_LOGIN_FAILED	このエラーは、ディレクトリ ログインに失敗し、ディレクトリが初期化されないことを示します。
DIRECTORY_LOGIN_NOT_ALLOWED	このエラーは、ディレクトリ ログインに失敗したことを示します。
DIRECTORY_TEMPORARY_UNAVAILABLE	このエラーは、ディレクトリが一時的に利用不能になっていることを示します。

エラー コード：	説明
EXISTING_FIRSTPARTY	このエラーは、すでにデバイス制御メディアが存在していることを示します。
HOLDFAILED	このエラーは、保留が回線制御層またはコール制御層によって拒否されたことを示します。
ILLEGAL_CALLINGPARTY	このエラーは、デバイス上にない発側を使用して、発呼しようとしたことを示します。
ILLEGAL_CALLSTATE	このエラーは、要求を呼び出すには、回線が正しい状態にないことを示します。
ILLEGAL_HANDLE	このエラーは、ハンドルが無効であることを示します。
ILLEGAL_MESSAGE_FORMAT	このエラーは、QBE プロトコル エラーが存在することを示します。
INCOMPATIBLE_PROTOCOL_VERSION	このエラーは、JTAPI と CTI のバージョンに互換性がないことを示します。CTI エラー プロトコルのバージョンをサポートしません。
INVALID_LINE_HANDLE	このエラーは、無効な回線ハンドルで回線操作を実行しようとしたことを示します。
INVALID_RING_OPTION	このエラーは、呼出音オプションが無効であることを示します。
LINE_GREATER_THAN_MAX_LINE	このエラーは、回線がこのデバイス上で利用できる最大回線数よりも多いことを示します。
LINE_INFO_DOES_NOT_EXIST	このエラーは、回線情報がデータベースに存在しないことを示します。
LINE_NOT_PRIMARY	このエラーは、コール制御から内部エラーが返されたことを示します。
LINECONTROL_FAILURE	このエラーは、新規のコールの状態が原因で、回線制御がそのコールの開始を拒否したことを示します。
MAX_NUMBER_OF_CTI_CONNECTIONS_REACHED	CTI 接続の最大数に達しました。
MSGWAITING_DESTN_INVALID	このエラーは、無効な DN に対してメッセージ待ちランプをセットしようとしたことを示します。メッセージ待ち宛先が見つかりません。
NO_ACTIVE_DEVICE_FOR_THIRDPARTY	このエラーは、サード パーティに対してアクティブなデバイスがないことを示します。
NO_CONFERENCE_BRIDGE	このエラーは、会議ブリッジがないことを示します。
NOT_INITIALIZED	このエラーは、CTI の初期化が完了する前に、プロバイダーをオープンしようとしたことを示します。
PROTOCOL_TIMEOUT	コール制御から内部エラーが返されました。
PROVIDER_ALREADY_OPEN	このエラーは、プロバイダーを再オープンしようとしたことを示します。
PROVIDER_CLOSED	このエラーは、すでにクローズされているプロバイダーをクローズしようとしたことを示します。
PROVIDER_NOT_OPEN	このエラーは、デバイスの一覧が不完全であるか、デバイスの一覧の照会がタイムアウトになったか、または照会が中断されたことを示します。

エラー コード :	説明
REDIRECT_CALL_CALL_TABLE_FULL	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALL_DESTINATION_BUSY	このエラーは、転送先が通話中であることを示します。
REDIRECT_CALL_DESTINATION_OUT_OF_ORDER	このエラーは、リダイレクト先が故障中ことを示します。
REDIRECT_CALL_DIGIT_ANALYSIS_TIMEOUT	このエラーは、ディジット分析がタイムアウトになったことを示します。これはコール制御から返された内部エラーです。
REDIRECT_CALL_DOES_NOT_EXIST	このエラーは、存在しないまたは現在有効ではないコールをリダイレクトしようとしたことを示します。
REDIRECT_CALL_INCOMPATIBLE_STATE	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALL_MEDIA_CONNECTION_FAILED	このエラーは、メディア接続エラーを示します。これはコール制御から返された内部エラーです。
REDIRECT_CALL_NORMAL_CLEARING	このエラーは、通常のコールのクリアのためにリダイレクトが失敗したことを示します。
REDIRECT_CALL_ORIGINATOR_ABANDONED	このエラーは、リダイレクトされるコールの遠端がハング アップしていることを示します。
REDIRECT_CALL_PARTY_TABLE_FULL	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALL_PENDING_REDIRECT_TRANSACTION	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALL_PROTOCOL_ERROR	このエラーは、プロトコル エラーを示します。これはコール制御から返された内部エラーです。
REDIRECT_CALL_UNKNOWN_DESTINATION	このエラーは、不明な宛先にリダイレクトしようとしたことを示します。
REDIRECT_CALL_UNKNOWN_ERROR	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALL_UNKNOWN_PARTY	このエラーは、不明な通話者が検出されたことを示します。これはコール制御から返された内部エラーです。
REDIRECT_CALL_UNRECOGNIZED_MANAGER	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_CALLINFO_ERR	このエラーは、コール制御から内部エラーが返されたことを示します。
REDIRECT_ERR	このエラーは、コール制御から内部エラーが返されたことを示します。
RETRIEVEFAILED	このエラーは、コールの取得が回線制御またはコール制御によって拒否されたことを示します。
RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	このエラーは、回線に他のコールがすでに存在するために、保留コールの取得中にエラーが発生したことを示します。
SSAPI_NOT_REGISTERED	このエラーは、内部サポート インターフェイスが初期化されていないときに、リダイレクト コマンドが発行されたことを示します。CTI が初期化を完了していないか、内部エラーが生じています。

エラー コード：	説明
TIMEOUT	このエラーは、要求がタイムアウトになったことを示します。
TRANSFER_INACTIVE	このエラーは、コンサルト転送が存在しないときに、転送を実行しようとしたことを示します。
TRANSFERFAILED	このエラーは、コール レッグの 1 つが遠端でハングアップしているか、接続解除されたために、転送が失敗した可能性があることを示します。
TRANSFERFAILED_CALLCONTROL_TIMEOUT	このエラーは、転送の間、コール制御から予期される応答を受信していないことを示します。
TRANSFERFAILED_DESTINATION_BUSY	このエラーは、使用中状態の宛先に転送しようとしたことを示します。
TRANSFERFAILED_DESTINATION_UNALLOCATED	このエラーは、登録されていない電話番号に転送しようとしたことを示します。
TRANSFERFAILED_OUTSTANDING_TRANSFER	このエラーは、既存の転送が進行中であることを示します。
UNDEFINED_LINE	このエラーは、指定された回線が、デバイスで見つからないことを示します。
UNKNOWN_GLOBAL_CALL_HANDLE	このエラーは、グローバル コール ハンドルが不明であることを示します。
UNRECOGNIZABLE_PDU	このエラーは、QBE プロトコル エラーが存在することを示します。
UNSPECIFIED	このエラーは、詳細不明のエラーが発生したことを示します。

CiscoEventIDs

このセクションでは、次のイベントについて説明します。

プロバイダー イベント

イベント名	イベント番号
CiscoProvFeatureUnRegisteredEv	0x40000008
CiscoRestrictedEv	0x40000009
CiscoAddrRestrictedEv	0x40000010
CiscoTermRestrictedEv	0x40000011
CiscoAddrActivatedEv	0x40000012
CiscoTermActivatedEv	0x40000013
CiscoAddrActivatedOnTerminalEv	0x40000014
CiscoAddrRestrictedOnTerminalEv	0x40000015
CiscoProviderCapabilityChangedEv	0x40000016
CiscoProvTerminalCapabilityChangedEv	0x40000017

端末イベント

イベント名	イベント番号
CiscoTermCreatedEv	0x40001001
CiscoTermDataEv	0x40001002
CiscoTermInServiceEv	0x40001003
CiscoTermOutOfServiceEv	0x40001004
CiscoTermRemovedEv	0x40001005
CiscoTermDeviceActiveStatusEv	0x40001006
CiscoTermDeviceAlertingStatusEv	0x40001007
CiscoTermDeviceHoldStatusEv	0x40001008
CiscoTermDeviceIdleStatusEv	0x40001009
CiscoTermButtonPressedEv	0x40001010
CiscoTermRegistraionFailedEv	0x40001011

イベント名	イベント番号
CiscoTermDNDStatusChangedEv	0x40001014
CiscoTermDeviceStateWhisperEv	0x40001015
CiscoTermDNDOptionChangedEv	0x40001016

アドレス イベント

イベント名	イベント番号
CiscoAddrCreatedEv	0x40002001
CiscoAddrInServiceEv	0x40002002
CiscoAddrOutOfServiceEv	0x40002003
CiscoAddrRemovedEv	0x40002004
CiscoOutOfServiceEv	0x40002005
CiscoAddrAddedToTerminalEv	0x40002006
CiscoAddrRemovedFromTerminalEv	0x40002007
CiscoAddrAutoAcceptStatusChangedEv	0x40002008
CiscoAddrIntercomInfoChangedEv	0x40002009
CiscoAddrIntercomInfoRestorationFailedEv	0x40002010
CiscoAddrRecordingConfigChangedEv	0x40002011
CiscoAddrParkStatusEv	0x40002012

コール イベント

イベント名	イベント番号
CiscoProvCallParkEv	0x40003001
CiscoConferenceEndEv	0x40003002
CiscoConferenceStartEv	0x40003003
CiscoConsultCallActiveEv	0x40003004
CiscoTransferEndEv	0x40003005
CiscoTransferStartEv	0x40003006
CiscoToneChangedEv	0x40003007
CiscoCallChangedEv	0x40003008
CiscoConferenceChainAddedEv	0x40003009
CiscoConferenceChainRemovedEv	0x40003010
CiscoCallSecurityStatusChangedEv	0x40003011

RTP イベント

イベント名	イベント番号
CiscoRTPInputStartedEv	0x40004001
CiscoRTPInputStoppedEv	0x40004002
CiscoRTPOutputStartedEv	0x40004003
CiscoRTPOutputStoppedEv	0x40004004
CiscoMediaOpenLogicalChannelEv	0x40004005
CiscoRTPInputKeyEv	0x40004006
CiscoRTPOutputKeyEv	0x40004007

TermConn イベント

イベント名	イベント番号
CiscoTermConnPrivacyChangedEv	0x40005001
CiscoCallCtlTermConnHeldReversionEv	0x40005002
CiscoTermConnSelectChangedEv	0x40005003
CiscoTermConnRecordingStartEv	0x40005004
CiscoTermConnRecordingEndEv	0x40005005
CiscoTermConnMonitoringStartEv	0x40005006
CiscoTermConnMonitoringEndEv	0x40005007
CiscoTermConnRecordingTargetInfoEv	0x40005008
CiscoTermConnMonitorInitiatorInfoEv	0x40005009
CiscoTermConnMonitorTargetInfoEv	0x4000500A

原因コード

以下のコードは CiscoFeatureReason インターフェイスで定義されています。

原因コード名	原因コード
REASON_TRANSFER	2
REASON_FORWARDNOANWSER	3
REASON_FORWARDBUSY	4
REASON_FORWARDALL	5
REASON_REDIRECT	6
REASON_BLINDTRANSFER	7
REASON_CONFERENCE	9
REASON_PARK	10
REASON_CALLPICKUP	11

原因コード名	原因コード
REASON_NORMAL	12
REASON_PARKREMINDER	15
REASON_UNPARK	16
REASON_BARGE	20
REASON_IMMDIVERT	21
REASON_FAC_CMC	22
REASON_QSIG_PR	23
REASON_REFER	24
REASON_REPLACE	25
REASON_CCM_REDIRECTION	26
REASON_DPARK_CALLPARK	27
REASON_DPARK_REVERSION	28
REASON_DPARK_UNPARK	29
REASON_SILENTMONITORING	31
REASON_MOBILITY	33
REASON_MOBILITY_IVR	34
REASON_MOBILITY_CELLPICKUP	35
REASON_MOBILITY_HANDIN	36
REASON_MOBILITY_HANDOUT	37
REASON_MOBILITY_FOLLOWME	38
REASON_CLICK_TO_CONFERENCE	39
REASON_FORWARD_NO_RETRIEVE	40

原因コード

原因コード名	原因コード
CAUSE_NOERROR	0X00 (0)
CAUSE_UNALLOCATEDNUMBER	0X01 (1)
CAUSE_NOROUTETOTRANSITNET	0X02 (2)
CAUSE_NOROUTETODDESTINATION	0X03 (3)
CAUSE_CHANUNACCEPTABLE	0X06 (6)
CAUSE_CALLBEINGDELIVERED	0X07 (7)
CAUSE_CTIPREEMPTNOREUSE	0X08 (8)
CAUSE_CTIPREEMPTFORREUSE	0X09 (9)
CAUSE_NORMALCALLCLEARING	0X10 (16)
CAUSE_USERBUSY	0X11 (17)
CAUSE_NOUSERRESPONDING	0X12 (18)
CAUSE_NOANSWERFROMUSER	0X13 (19)

原因コード名	原因コード
CAUSE_SUBSCRIBERABSENT	0X14 (20)
CAUSE_CALLREJECTED	0X15 (21)
CAUSE_NUMBERCHANGED	0X16 (22)
CAUSE_EXCHANGEROUTINGERROR	0X19 (25)
CAUSE_NONSELECTEDUSERCLEARING	0X1A (26)
CAUSE_DDESTINATIONOUTOFORDER	0X1B (27)
CAUSE_INVALIDNUMBERFORMAT	0X1C (28)
CAUSE_FACILITYREJECTED	0X1D (29)
CAUSE_RESPONSETOSTATUSENQUIRY	0X1E (30)
CAUSE_NORMALUNSPECIFIED	0X1F (31)
CAUSE_NOCIRCAVAIL	0X22 (34)
CAUSE_NETOUTOFORDER	0X26 (38)
CAUSE_TEMPORARYFAILURE	0X29 (41)
CAUSE_SWITCHINGEQUIPMENT CONGESTION	0X2A (42)
CAUSE_ACCESSINFORMATION DISCARDED	0X2B (43)
CAUSE_REQCIRCNAVIL	0X2C (44)
CAUSE_CTIPRECEDENCECALLBLOCKED	0X2E (46)
CAUSE_RESOURCESNAVAIL	0X2F (47)
CAUSE_QUALOFSERVNAVAIL	0X31 (49)
CAUSE_REQFACILITYNOTSUBSCRIBED	0X32 (50)
CAUSE_SERVOPERATIONVIOLATED	0X35 (53)
CAUSE_INCOMINGCALLBARRED	0X36 (54)
CAUSE_BCNAUTHORIZED	0X39 (57)
CAUSE_BCBPRESENTLYAVAIL	0X3A (58)
CAUSE_SERVNOTAVAILUNSPECIFIED	0X3F (63)
CAUSE_BEARERCAPNIMPL	0X41 (65)
CAUSE_CHANTYPENIMPL	0X42 (66)
CAUSE_REQFACILITYNIMPL	0X45 (69)
CAUSE_ONLYRDIVEARERCAPAVAIL	0X46 (70)
CAUSE_SERVOROPTNAVAILORIMPL	0X4F (79)
CAUSE_INVALIDCALLREFVALUE	0X51 (81)
CAUSE_IDENTIFIEDCHANDOESNOTEXIST	0X52 (82)
CAUSE_SUSPCALLBUTNOTTHISONE	0X53 (83)
CAUSE_CALLIDINUSE	0X54 (84)
CAUSE_NOCALLSUSPENDED	0X55 (85)
CAUSE_REQCALLIDHASBEENCLEARED	0X56 (86)
CAUSE_INCOMPATABLEDDESTINATION	0X58 (88)
CAUSE_DESTNUMMISSANDDCNOTSUB	0X5A (90)

原因コード名	原因コード
CAUSE_INVALIDTRANSITNETSEL	0X5B (91)
CAUSE_INVALIDMESSAGEUNSPECIFIED	0X5F (95)
CAUSE_MANDATORYIEMISSING	0X60 (96)
CAUSE_MSGTYPENIMPL	0X61 (97)
CAUSE_MSGTYPENCOMPATWCS	0X62 (98)
CAUSE_IENIMPL	0X63 (99)
CAUSE_INVALIDIECONTENTS	0X64 (100)
CAUSE_MSGNCOMPATABLEWCS	0X65 (101)
CAUSE_RECOVERYONTIMEREXPIRY	0X66 (102)
CAUSE_PROTOCOLERRORUNSPECIFIED	0X6F (111)
CAUSE_CTIPRECEDENCELEVEL EXCEEDED	0X7A (122)
CAUSE_CTIDEVICENOTPREEMPTABLE	0X7B (123)
CAUSE_OUTOFBANDWIDTH	0X7D (125)
CAUSE_INTERWORKINGUNSPECIFIED	0X7F (127)
CAUSE_CTIPRECEDENCEOUTOF BANDWIDTH	0X81 (129)
CAUSE_REDIRECTED	0XC9 (200)
CAUSE_INTERNALCAUSE	0X1F4 (500)
CAUSE_OUTBOUND_TRANSFER	0X1F5 (501)
CAUSE_OUTBOUND_CONFERENCE	0X1F6 (502)
CAUSE_INBOUND_TRANSFER	0X1F7 (503)
CAUSE_INBOUND_CONFERENCE	0X1F8 (504)
CAUSE_INBOUND_BLINDTRANSFER	0X1F9 (505)
CAUSE_CTIMANAGER_FAILURE	0x1FB (507)
CAUSE_CALLMANAGER_FAILURE	0x1FC (508)
CAUSE_BARGE	0x1FD(509)
CAUSE_FAC_CMC	0x1FE (510)
CAUSE_QSIG_PR	0x1FF (511)
CAUSE_DPARK	0x200 (512)
CAUSE_DPARK_UNPARK	0x201 (513)
CAUSE_DPARK_REMINDER	0x202 (514)
CAUSE_QUIET_CLEAR	0x203 (515)
CAUSE_CTICONFERENCEFULL	0X40000 + CAUSE_NOERROR
CAUSE_CALLSPLIT	0X60000 + CAUSE_NOERROR
CAUSE_CTIDROPCONFEREE	0X70000 + CAUSE_NOERROR
CAUSE_CTICCMSIP400BADREQUEST	0X1000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP401UNAUTHORIZED	0X2000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP402PAYMENT REQUIRED	0X3000000 + CAUSE_CALLREJECTED

原因コード名	原因コード
CAUSE_CTICCMSIP403FORBIDDEN	0X4000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP404NOTFOUND	0X5000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP405METHODNOT ALLOWED	0X6000000 + CAUSE_SERVNOTAVAILUNSPECIFIED
CAUSE_CTICCMSIP406NOTACCEPTABLE	0X7000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP407PROXYAUTHENTIC ATIONREQUIRED	0X8000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP408REQUESTTIMEOUT	0X9000000 + CAUSE_RECOVERYONTIMEREXPIRY
CAUSE_CTICCMSIP410GONE	0XB000000 + CAUSE_NUMBERCHANGED
CAUSE_CTICCMSIP411LENGTHREQUIRED	0XC000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP413REQUESTENTITY TOOLONG	0XE000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP414REQUESTURI TOOLONG	0XF000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP415UNSUPPORTED MEDIATYPE	0X10000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP416UNSUPPORTED URISCHEME	0X11000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP420BADEXTENSION	0X15000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP421EXTENSTION REQUIRED	0X16000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP423INTERVALTOO BRIEF	0X18000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP480TEMPORARILY UNAVAILABLE	0X40000000 + CAUSE_NOUSERRESPONDING
CAUSE_CTICCMSIP481CALLLEGDOESNOT EXIST	0X41000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP482LOOPDETECTED	0X42000000 + CAUSE_EXCHANGEROUTINGERROR
CAUSE_CTICCMSIP483TOOMANYHOOPS	0X43000000 + CAUSE_EXCHANGEROUTINGERROR
CAUSE_CTICCMSIP484ADDRESS INCOMPLETE	0X44000000 + CAUSE_INVALIDNUMBERFORMAT
CAUSE_CTICCMSIP485AMBIGUOUS	0X45000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP486BUSYHERE	0X46000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP487REQUEST TERMINATED	0X47000000 + CAUSE_NORMALUNSPECIFIED

原因コード名	原因コード
CAUSE_CTICCMSIP488NOTACCEPTABLE HERE	0X48000000 + CAUSE_NORMALUNSPECIFIED
CAUSE_CTICCMSIP491REQUESTPENDING	0X4B000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP493UNDECIPHERABLE	0X4D000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP500SERVERINTERNALE RROR	0X54000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP501NOTIMPLEMENTED	0X55000000 + CAUSE_SERVOROPTNAVAILORIMPL
CAUSE_CTICCMSIP502BADGATEWAY	0X56000000 + CAUSE_NETOUTOFORDER
CAUSE_CTICCMSIP503SERVICE UNAVAILABLE	0X57000000 + CAUSE_TEMPORARYFAILURE
CAUSE_CTICCMSIP504SERVERTIMEOUT	0X58000000 + CAUSE_RECOVERYONTIMEREXPIRY
CAUSE_CTICCMSIP505SIPVERSIONNOT SUPPORTED	0X59000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP513MESSAGETOO LARGE	0X5A000000 + CAUSE_INTERWORKINGUNSPECIFIED
CAUSE_CTICCMSIP600BUSY EVERYWHERE	0XA1000000 + CAUSE_USERBUSY
CAUSE_CTICCMSIP603DECLINE	0XA2000000 + CAUSE_CALLREJECTED
CAUSE_CTICCMSIP604DOESNOTEXIST ANYWHERE	0XA3000000 + CAUSE_UNALLOCATEDNUMBER
CAUSE_CTICCMSIP606NOTACCEPTABLE	0XA4000000 + CAUSE_NORMALUNSPECIFIED

その他のトラブルシューティング情報

JTAPI デバッグ出力の表示

JTAPI デバッグ出力を表示するには、JTPREFS アプリケーションを使用してトレース設定を変更します。JTPREFS アプリケーションを使用すると、さまざまなトレースを有効または無効にできます。

JTPREFS は JTAPI クラスとともに **%SystemRoot%¥java¥lib** ディレクトリにインストールされています。デフォルトでは、Cisco JTAPI Preferences は **Program Files¥JTAPITools** にインストールされています。

Cisco JTAPI の初期設定ユーティリティを開くには、[スタート] > [プログラム] > [Cisco JTAPI] > [JTAPI Preferences] の順に選択してください。

次のトレース レベルが定義されています。

- WARNING：警告イベント
- INFORMATIONAL：ステータス イベント
- DEBUG：デバッグ イベント

DEBUG が有効になっている場合、JTPREFS を使用して、さまざまなデバッグ レベルを有効または無効にできます。

次のデバッグ レベルが定義されています。

- TAPI_DEBUGGING：JTAPI のメソッドおよびイベントのトレース
- TAPI_IMPLDEBUGGING：内部 JTAPI 実装トレース
- CTI_DEBUGGING：JTAPI 実装に送信される Cisco Unified Communications Manager イベントのトレース
- CTIIMPL_DEBUGGING：内部 CTICLIENT 実装トレース
- PROTOCOL_DEBUGGING：CTI プロトコルの完全なデコード
- MISC_DEBUGGING：各種の低レベル デバッグ トレース

トレースは、デフォルトのアプリケーション ディレクトリとは異なる特定のパスとフォルダにリダイレクトできます。JTAPI の連続した起動または 1 つ以上の同時起動の間で、同じトレース フォルダを使用できます。JTAPI の起動ごとにトレースを異なるフォルダへ送ることもできます。これにより、JTAPI の同時インスタンスごとに独立したトレース保存先を設定できます。

JTAPI クライアント インストーラのログ ファイル

インストールおよびアンインストールのプロセス中に発生するエラーを検出するため、2 つのログ ファイルが生成されます。このファイルはいずれも、インストーラの実行元となる場所に作成されます。

- ismpInstall.log：インストール中のイベントを記録する
- ismpUninstall.log：アンインストール中のイベントを記録する

エラー メッセージには、インストール手順の一部として実行されたウィザード ビーンに関する情報のほか、例外があればそれも含まれます。

ISMP インストーラに関するトラブルシューティングのヒント

番号	問題の説明	原因	解決策
1	ISMP のアンインストールで、目的のインストール済みディレクトリが削除されない。	アンインストーラが起動されたディレクトリ	アンインストーラは少なくとも目的のインストール ディレクトリの 1 レベル上から起動する必要があります。
2	インストール時に適切な言語で詳細が表示されない。	ロケール ファイルが正しくない。	サポート担当者にこの問題をただちに報告して、メッセージの変更またはエラーを知らせてください。
3	インストーラまたはアンインストーラがエラーをスローする。	JVM が削除されたか、または互換性のないバージョンに置換された。	インストーラにはビルトインの JVM が備えられており、マシンに JVM が存在しない場合はこれが同時にインストールされます。このエラーが発生した場合は、ファイルを手動で削除する必要があります。

4	インストーラは正常に実行されたが、ファイルがコピーされていない。	権限	コピー先フォルダに対する適切な書き込み権限があることを確認してください。この問題は UNIX プラットフォームで発生する場合があります。
5	インストーラまたはアンインストーラが、インストール処理中に例外をスローするかクラッシュする。	バージョン名またはフォルダ名の問題	生成されるログ ファイルを参照して、エラーの原因となった手順を判断してください。
6	アップグレード バージョンのインストール時に「アップグレード」メッセージが表示されない。	.jtapiver.ini がない	このファイルに、現在の jtapi インストールの詳細が記録されています。このファイルを誤って削除してしまうと、アップグレードまたは再インストールの実行中の表示に問題が発生します。アップグレード/再インストールまたはダウングレードが失敗する場合は、ユーザが .jtapi/bin フォルダおよび .jtapi/lib フォルダからファイルを手動で削除して、次回のインストールが成功するよう再度インストーラを実行する必要があります。

ログイン タイムアウトによりプロバイダーを作成できない

このエラーは、CTI から ProviderOpenRequest に対する認証応答がない場合に発生します。エラーの原因として、次の場合があります。

- LDAP 接続性の問題
- データベース遅延
- その他の原因で CTIManager がビジーであるため、要求に応じることができない。

解決策は、アプリケーションの再試行です。再試行しても ProviderOpenRequest が失敗する場合は、接続先の設定やログイン ID、パスワードが正しいか確認してください。また、CTIManager サービスが正常に動作しているかを確認してください。

APPENDIX D

リリースごとの Cisco Unified JTAPI オペレーション

表 D-1 は、Cisco Unified Communications Manager リリースにおける Cisco Unified JTAPI の機能の一覧が示されています。「サポート対象」、「サポート対象外」、「変更」、「検討中または調査中」として表します。詳細については、http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html で入手可能な『Cisco Unified Communications Manager JTAPI Developers Guide』を参照してください。

表の凡例：

✓：サポート対象、✗：サポート対象外、ⓘ：変更、UCR：検討中または調査中

表 D-1　Cisco Unified Communications Manager リリースによる JTAPI の機能

JTAPI の機能	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1.2	7.1.3 (UCR)	8.0 (UCR)
耐障害性に対する CTI Manager または CTI サポート	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cisco CallManager エクステンション モビリティのサポート	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ブラインド転送（リダイレクト使用）	✓	✓	✓	ⓘ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
転送のサポート	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
リダイレクトを使用した、元の着信者のリセット	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CiscoAddrInServiceEv または CiscoAddrOutOFServiceEv	ⓘ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ローカリゼーションおよびインターナショナリゼーション	✗	✓	✓	ⓘ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ディレクトリからのユーザの削除	✗	✗	ⓘ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
パークおよびパーク解除	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

表 D-1　Cisco Unified Communications Manager リリースによる JTAPI の機能 （続き）

JTAPI の機能	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1.2	7.1.3 (UCR)	8.0 (UCR)
コールパーク番号の監視	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
コール理由の拡張機能	✗	✗	ℹ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
デバイス データ パススルー	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cisco JTAPI の自動インストール	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DN あたりの複数コール	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
共用回線のサポート	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
転送	✓	✓	✓	ℹ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
直接転送	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
会議	✓	✓	✓	ℹ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
参加	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
プライバシー リリース	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
割り込みと C 割り込み	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ダイナミック ポート登録	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ルート ポイントでのメディア終端	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ボイスメールへの転送	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
発信側番号の変更	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
プレゼンテーション表示のサポート	✗	✗	✗	✓	✓	✓	✓	✓	ℹ	✓	✓	✓	✓	✓	✓
QSIG-PR	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FAC および CMC のサポート	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
デバイス ステート サーバ	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
スーパープロバイダー機能	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows 2003 のサポート	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ダイレクト コール パーク	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
帯域幅不足および未登録時の転送	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

表 D-1　Cisco Unified Communications Manager リリースによる JTAPI の機能 （続き）

JTAPI の機能	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1.2	7.1.3 (UCR)	8.0 (UCR)
ボイス メールボックスのサポート	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Privacy On Hold	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
保留の復帰（4.2.1.SR1）	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MLPP（4.2.2）のサポート	✓	✓	✓	✓	✓	ℹ	✓	✓	✓	✓	✓	✓	✓	✓	✓
会議に関する拡張：コントローラ以外による会議への参加者の追加（4.2.2）	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
会議のチェーニング（4.2.2）	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
CiscoRTPHandle インターフェイス（4.2.2）	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
CiscoTermRegistrationFailedEv：新しいエラー コード（4.2.3）	✗	✗	✗	✓	✓	ℹ	✓	ℹ	✓	✓	✓	✓	✓	✓	✓
ネットワーク イベント	✓	✓	✓	✓	✓	✓	✓	ℹ	✓	✓	✓	✓	✓	✓	✓
BWC の拡張	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
ヘアピン サポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Unicode のサポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
SRTP サポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
パーティションのサポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
セキュリティ（TLS）のサポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
代替スクリプトのサポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
SIP 機能の Refer および Replace	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
SIP のエンド ポイント サポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
スーパープロバイダーの変更通知およびコール パーク DN のモニタリング機能	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
3XX	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
コールの選択状態	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓

表 D-1　Cisco Unified Communications Manager リリースによる JTAPI の機能 （続き）

JTAPI の機能	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1.2	7.1.3 (UCR)	8.0 (UCR)
QoS のサポート	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Linux および Solaris のインストーラ	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
インターコム サポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
セキュア会議のサポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
モニタリングおよび録音	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
アラビア語とヘブライ語の言語サポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
Do Not Disturb（サイレント）のサポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
回線をまたいで参加（SCCP）	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓
証明書のダウンロード API の機能拡張	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
エクステンション モビリティのインターコム サポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
回線をまたいで参加（SIP フォンのサポート）	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
ロケール インフラストラクチャの拡張	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
DND コール拒否（DND-R）	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
発信者の正規化（CPN）	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
クリック ツー会議	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
IPv6 のサポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Windows Vista のサポート	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EMLogin ユーザ名 API	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
CiscoJTAPIPeer の SetJtapiProperties API	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
会議から参加者をドロップする	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
スワップ / キャンセル - 転送 / 会議の動作の変更	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
回線をまたいだ直接転送	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓

表 D-1　Cisco Unified Communications Manager リリースによる JTAPI の機能 （続き）

JTAPI の機能	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	6.0	6.1	7.0	7.1.2	7.1.3 (UCR)	8.0 (UCR)
パーク モニタリングの機能拡張	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Assisted DPark	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
拡張された MWI	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
論理パーティション設定	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
ロールオーバーのサポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Address と Terminal の設定 （最大コール数、ボイス メール、ビジー トリガなど）	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
エンド ツー エンド コールのトレース	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
クラスタ間のエクステンション モビリティ	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
ハント リストのサポート	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

APPENDIX E

CTI でサポートされるデバイス

表 E-1 は CTI でサポートされるデバイスの情報の一覧です。

表の凡例：

: サポート対象、: サポート対象外、: 変更、UCR：検討中または調査中

表 E-1　　CTI でサポートされるデバイスの一覧

デバイス / 電話機のモデル	SCCP	SIP	コメント
アナログ電話機			
Cisco 12 S			
Cisco 12 SP			
Cisco 12 SP			
Cisco 30 SP			
Cisco 30 VIP			
Cisco 3911			CTI でサポートされるデバイスではない
Cisco 7902			
Cisco 7905			
Cisco 7906			
Cisco 7910			
Cisco 7911			
Cisco 7912			
Cisco 7914 Sidecar			
Cisco 7915 Sidecar	**UCR**	**UCR**	
Cisco 7916 Sidecar	**UCR**	**UCR**	
Cisco 7920			

表 E-1　CTI でサポートされるデバイスの一覧 （続き）

デバイス / 電話機のモデル	SCCP	SIP	コメント
Cisco 7921	✓	✗	
Cisco 7931	✓	✗	ロールオーバーが無効の場合だけ CTI でサポートされる
Cisco 7935	✓	✗	
Cisco 7936	✓	✗	
Cisco 7937	✓	✗	
Cisco 7940	✓	✗	
Cisco 7941	✓	✓	
Cisco 7941 G-GE	✓	✓	
Cisco 7942	✓	✓	
Cisco 7945	✓	✓	
Cisco 7960	✓	✗	
Cisco 7961	✓	✓	
Cisco 7961G-GE	✓	✓	
Cisco 7962	✓	✓	
Cisco 7965	✓	✓	
Cisco 7970	✓	✓	
Cisco 7971	✓	✓	
Cisco 7975	✓	✓	
Cisco 7985	✓	✗	
Cisco ATA	✓	✗	
Cisco IP Communicator	✓	✗	
Cisco Unified Personal Communicator	✗	✗	デスクトップ モードでの実行時に、物理デバイス次第で CTI でサポートされる。ソフトフォン モードは未テスト。
Cisco VGC Phone	✓	✗	
VG224	✓	✗	
VG248	✓	✗	

表 E-1　CTI でサポートされるデバイスの一覧 （続き）

デバイス / 電話機のモデル	SCCP	SIP	コメント
CTI ポート	—	—	SCCP または SIP を使用しない仮想デバイスを CTI でサポート
CTI ルート ポイント	—	—	SCCP または SIP を使用しない仮想デバイスを CTI でサポート
CTI ルート ポイント（パイロット ポイント）	—	—	SCCP または SIP を使用しない仮想デバイスを CTI でサポート
ISDN BRI Phone	—	—	CTI でサポートされるデバイスではない

APPENDIX F

定数フィールド値

この付録では、static final フィールドとその値を示します。

com.cisco.*

CiscoAddrActivatedEv

com.cisco.jtapi.extensions.CiscoAddrActivatedEv		
public static final int	ID	1073741842

CiscoAddrActivatedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrActivatedOnTerminalEv		
public static final int	ID	1073741844

CiscoAddrAddedToTerminalEv

com.cisco.jtapi.extensions.CiscoAddrAddedToTerminalEv		
public static final int	ID	1073750022

CiscoAddrAutoAcceptStatusChangedEv

com.cisco.jtapi.extensions.CiscoAddrAutoAcceptStatusChangedEv		
public static final int	ID	1073750024

CiscoAddrCreatedEv

com.cisco.jtapi.extensions.CiscoAddrCreatedEv		
public static final int	ID	1073750017

CiscoAddress

com.cisco.jtapi.extensions.CiscoAddress		
public static final int	APPLICATION_CONTROLLED_RECORDING	2
public static final int	AUTO_RECORDING	1
public static final int	AUTOACCEPT_OFF	0
public static final int	AUTOACCEPT_ON	1
public static final int	AUTOANSWER_OFF	0
public static final int	AUTOANSWER_UNKNOWN	3
public static final int	AUTOANSWER_WITHHEADSET	1
public static final int	AUTOANSWER_WITHSPEAKERSET	2
public static final int	EXTERNAL	2
public static final int	EXTERNAL_UNKNOWN	3
public static final int	IN_SERVICE	1
public static final int	INTERNAL	1
public static final int	MONITORING_TARGET	5
public static final int	NO_RECORDING	0
public static final int	OUT_OF_SERVICE	0
public static final int	RINGER_DEFAULT	0
public static final int	RINGER_DISABLE	1
public static final int	RINGER_ENABLE	2
public static final int	UNKNOWN	4

CiscoAddrInServiceEv

com.cisco.jtapi.extensions.CiscoAddrInServiceEv		
public static final int	ID	1073750018

CiscoAddrIntercomInfoChangedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoChangedEv		
public static final int	ID	1073750025

CiscoAddrIntercomInfoRestorationFailedEv

com.cisco.jtapi.extensions.CiscoAddrIntercomInfoRestorationFailedEv		
public static final int	ID	1073750032

CiscoAddrOutOfServiceEv

com.cisco.jtapi.extensions.CiscoAddrOutOfServiceEv		
public static final int	ID	1073750019

CiscoAddrRecordingConfigChangedEv

com.cisco.jtapi.extensions.CiscoAddrRecordingConfigChangedEv		
public static final int	ID	1073750033

CiscoAddrRemovedEv

com.cisco.jtapi.extensions.CiscoAddrRemovedEv		
public static final int	ID	1073750020

CiscoAddrRemovedFromTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRemovedFromTerminalEv		
public static final int	ID	1073750023

CiscoAddrRestrictedEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedEv		
public static final int	ID	1073741840

CiscoAddrRestrictedOnTerminalEv

com.cisco.jtapi.extensions.CiscoAddrRestrictedOnTerminalEv		
public static final int	ID	1073741845

CiscoCall

com.cisco.jtapi.extensions.CiscoCall		
public static final int	CALLSECURITY_AUTHENTICATED	2
public static final int	CALLSECURITY_ENCRYPTED	3
public static final int	CALLSECURITY_NOTAUTHENTICATED	1
public static final int	CALLSECURITY_UNKNOWN	0
public static final int	FEATUREPRIORITY_EMERGENCY	3
public static final int	FEATUREPRIORITY_NORMAL	1
public static final int	FEATUREPRIORITY_URGENT	2
public static final int	PLAYTONE_BOTHLOCALANDREMOTE	2
public static final int	PLAYTONE_LOCALONLY	0
public static final int	PLAYTONE_NOLOCAL_OR_REMOTE	3
public static final int	PLAYTONE_REMOTEONLY	1
public static final int	SILENT_MONITOR	1

CiscoCallChangedEv

com.cisco.jtapi.extensions.CiscoCallChangedEv		
public static final int	ID	1073754120

CiscoCallCtlTermConnHeldReversionEv

com.cisco.jtapi.extensions.CiscoCallCtlTermConnHeldReversionEv		
public static final int	ID	1073762306

CiscoCallEv

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_ACCESSINFORMATIONDISCARDED	43
public static final int	CAUSE_BARGE	509
public static final int	CAUSE_BCBPRESENTLYAVAIL	58
public static final int	CAUSE_BCNAUTHORIZED	57
public static final int	CAUSE_BEARERCAPNIMPL	65
public static final int	CAUSE_CALLBEINGDELIVERED	7
public static final int	CAUSE_CALLIDINUSE	84
public static final int	CAUSE_CALLMANAGER_FAILURE	508
public static final int	CAUSE_CALLREJECTED	21
public static final int	CAUSE_CALLSPLIT	393216
public static final int	CAUSE_CHANTYPENIMPL	66
public static final int	CAUSE_CHANUNACCEPTABLE	6
public static final int	CAUSE_CTICCMSIP400BADREQUEST	16777257
public static final int	CAUSE_CTICCMSIP401UNAUTHORIZED	33554453
public static final int	CAUSE_CTICCMSIP402PAYMENTREQUIRED	50331669
public static final int	CAUSE_CTICCMSIP403FORBIDDEN	67108885
public static final int	CAUSE_CTICCMSIP404NOTFOUND	83886081
public static final int	CAUSE_CTICCMSIP405METHODNOTALLOWED	100663359
public static final int	CAUSE_CTICCMSIP406NOTACCEPTABLE	117440591
public static final int	CAUSE_CTICCMSIP407PROXYAUTHENTICATIONREQUIRED	134217749
public static final int	CAUSE_CTICCMSIP408REQUESTTIMEOUT	150995046
public static final int	CAUSE_CTICCMSIP410GONE	184549398
public static final int	CAUSE_CTICCMSIP411LENGTHREQUIRED	201326719
public static final int	CAUSE_CTICCMSIP413REQUESTENTITYTOOLONG	234881151

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_CTICCMSIP414REQUESTURITOOLONG	251658367
public static final int	CAUSE_CTICCMSIP415UNSUPPORTEDMEDIATYPE	268435535
public static final int	CAUSE_CTICCMSIP416UNSUPPORTEDURISCHEME	285212799
public static final int	CAUSE_CTICCMSIP420BADEXTENSION	352321663
public static final int	CAUSE_CTICCMSIP421EXTENSTIONREQUIRED	369098879
public static final int	CAUSE_CTICCMSIP423INTERVALTOOBRIEF	402653311
public static final int	CAUSE_CTICCMSIP480TEMPORARILYUNAVAILABLE	1073741842
public static final int	CAUSE_CTICCMSIP481CALLLEGDOESNOTEXIST	1090519081
public static final int	CAUSE_CTICCMSIP482LOOPDETECTED	1107296281
public static final int	CAUSE_CTICCMSIP483TOOMANYHOOPS	1124073497
public static final int	CAUSE_CTICCMSIP484ADDRESSINCOMPLETE	1140850716
public static final int	CAUSE_CTICCMSIP485AMBIGUOUS	1157627905
public static final int	CAUSE_CTICCMSIP486BUSYHERE	1174405137
public static final int	CAUSE_CTICCMSIP487REQUESTTERMINATED	1191182367
public static final int	CAUSE_CTICCMSIP488NOTACCEPTABLEHERE	1207959583
public static final int	CAUSE_CTICCMSIP491REQUESTPENDING	1258291217
public static final int	CAUSE_CTICCMSIP493UNDECIPHERABLE	1291845649
public static final int	CAUSE_CTICCMSIP500SERVERINTERNALERROR	1409286185
public static final int	CAUSE_CTICCMSIP501NOTIMPLEMENTED	1426063439
public static final int	CAUSE_CTICCMSIP502BADGATEWAY	1442840614
public static final int	CAUSE_CTICCMSIP503SERVICEUNAVAILABLE	1459617833
public static final int	CAUSE_CTICCMSIP504SERVERTIMEOUT	1476395110
public static final int	CAUSE_CTICCMSIP505SIPVERSIONNOTSUPPORTED	1493172351

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_CTICCMSIP513MESSAGETOOLARGE	1509949567
public static final int	CAUSE_CTICCMSIP600BUSYEVERYWHERE	-1593835503
public static final int	CAUSE_CTICCMSIP603DECLINE	-1577058283
public static final int	CAUSE_CTICCMSIP604DOESNOTEXISTANYWHERE	-1560281087
public static final int	CAUSE_CTICCMSIP606NOTACCEPTABLE	-1543503841
public static final int	CAUSE_CTICONFERENCEFULL	262144
public static final int	CAUSE_CTIDEVICENOTPREEMPTABLE	123
public static final int	CAUSE_CTIDROPCONFEREE	458752
public static final int	CAUSE_CTIMANAGER_FAILURE	507
public static final int	CAUSE_CTIPRECEDENCECALLBLOCKED	46
public static final int	CAUSE_CTIPRECEDENCELEVELEXCEEDED	122
public static final int	CAUSE_CTIPRECEDENCEOUTOFBANDWIDTH	129
public static final int	CAUSE_CTIPREEMPTFORREUSE	9
public static final int	CAUSE_CTIPREEMPTNOREUSE	8
public static final int	CAUSE_DESTINATIONOUTOFORDER	27
public static final int	CAUSE_DESTNUMMISSANDDCNOTSUB	90
public static final int	CAUSE_DPARK	512
public static final int	CAUSE_DPARK_REMINDER	514
public static final int	CAUSE_DPARK_UNPARK	513
public static final int	CAUSE_EXCHANGEROUTINGERROR	25
public static final int	CAUSE_FAC_CMC	510
public static final int	CAUSE_FACILITYREJECTED	29
public static final int	CAUSE_IDENTIFIEDCHANDOESNOTEXIST	82
public static final int	CAUSE_IENIMPL	99
public static final int	CAUSE_INBOUNDBLINDTRANSFER	505
public static final int	CAUSE_INBOUNDCONFERENCE	504
public static final int	CAUSE_INBOUNDTRANSFER	503
public static final int	CAUSE_INCOMINGCALLBARRED	54
public static final int	CAUSE_INCOMPATABLEDDESTINATION	88
public static final int	CAUSE_INTERWORKINGUNSPECIFIED	127
public static final int	CAUSE_INVALIDCALLREFVALUE	81
public static final int	CAUSE_INVALIDIECONTENTS	100

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_INVALIDMESSAGEUNSPECIFIED	95
public static final int	CAUSE_INVALIDNUMBERFORMAT	28
public static final int	CAUSE_INVALIDTRANSITNETSEL	91
public static final int	CAUSE_MANDATORYIEMISSING	96
public static final int	CAUSE_MSGNCOMPATABLEWCS	101
public static final int	CAUSE_MSGTYPENCOMPATWCS	98
public static final int	CAUSE_MSGTYPENIMPL	97
public static final int	CAUSE_NETOUTOFORDER	38
public static final int	CAUSE_NOANSWERFROMUSER	19
public static final int	CAUSE_NOCALLSUSPENDED	85
public static final int	CAUSE_NOCIRCAVAIL	34
public static final int	CAUSE_NOERROR	0
public static final int	CAUSE_NONSELECTEDUSERCLEARING	26
public static final int	CAUSE_NORMALCALLCLEARING	16
public static final int	CAUSE_NORMALUNSPECIFIED	31
public static final int	CAUSE_NOROUTETODDESTINATION	3
public static final int	CAUSE_NOROUTETOTRANSITNET	2
public static final int	CAUSE_NOUSERRESPONDING	18
public static final int	CAUSE_NUMBERCHANGED	22
public static final int	CAUSE_ONLYRDIVEARERCAPAVAIL	70
public static final int	CAUSE_OUTBOUNDCONFERENCE	502
public static final int	CAUSE_OUTBOUNDTRANSFER	501
public static final int	CAUSE_OUTOFBANDWIDTH	125
public static final int	CAUSE_PROTOCOLERRORUNSPECIFIED	111
public static final int	CAUSE_QSIG_PR	511
public static final int	CAUSE_QUALOFSERVNAVAIL	49
public static final int	CAUSE_QUIET_CLEAR	515
public static final int	CAUSE_RECOVERYONTIMEREXPIRY	102
public static final int	CAUSE_REDIRECTED	200
public static final int	CAUSE_REQCALLIDHASBEENCLEARED	86
public static final int	CAUSE_REQCIRCNAVIL	44
public static final int	CAUSE_REQFACILITYNIMPL	69
public static final int	CAUSE_REQFACILITYNOTSUBSCRIBED	50
public static final int	CAUSE_RESOURCESNAVAIL	47
public static final int	CAUSE_RESPONSETOSTATUSENQUIRY	30

com.cisco.jtapi.extensions.CiscoCallEv		
public static final int	CAUSE_SERVNOTAVAILUNSPECIFIED	63
public static final int	CAUSE_SERVOPERATIONVIOLATED	53
public static final int	CAUSE_SERVOROPTNAVAILORIMPL	79
public static final int	CAUSE_SUBSCRIBERABSENT	20
public static final int	CAUSE_SUSPCALLBUTNOTTHISONE	83
public static final int	CAUSE_SWITCHINGEQUIPMENTCONGESTION	42
public static final int	CAUSE_TEMPORARYFAILURE	41
public static final int	CAUSE_UNALLOCATEDNUMBER	1
public static final int	CAUSE_USERBUSY	17

CiscoCallSecurityStatusChangedEv

com.cisco.jtapi.extensions.CiscoCallSecurityStatusChangedEv		
public static final int	ID	1073754129

CiscoConferenceChainAddedEv

com.cisco.jtapi.extensions.CiscoConferenceChainAddedEv		
public static final int	ID	1073754121

CiscoConferenceChainRemovedEv

com.cisco.jtapi.extensions.CiscoConferenceChainRemovedEv		
public static final int	ID	1073754128

CiscoConferenceEndEv

com.cisco.jtapi.extensions.CiscoConferenceEndEv		
public static final int	ID	1073754114

CiscoConferenceStartEv

com.cisco.jtapi.extensions.CiscoConferenceStartEv		
public static final int	ID	1073754115

CiscoConnection

com.cisco.jtapi.extensions.CiscoConnection		
public static final int	ADDRESS_SEARCH_SPACE	2
public static final int	CALLED_ADDRESS_DEFAULT	0
public static final int	CALLED_ADDRESS_SET_TO_PREFERREDCALLEDPARTY	2
public static final int	CALLED_ADDRESS_SET_TO_REDIRECT_DESTINATION	1
public static final int	CALLED_ADDRESS_UNCHANGED	0
public static final int	CALLINGADDRESS_SEARCH_SPACE	1
public static final int	DEFAULT_SEARCH_SPACE	0
public static final int	REASON_DIRECTCALL	1
public static final int	REASON_FORWARDALL	5
public static final int	REASON_FORWARDBUSY	4
public static final int	REASON_FORWARDNOANSWER	3
public static final int	REASON_OUTBOUND	99
public static final int	REASON_REDIRECT	6
public static final int	REASON_TRANSFERREDCALL	2
public static final int	REDIRECT_DROP_ON_FAILURE	1
public static final int	REDIRECT_NORMAL	2

CiscoConsultCallActiveEv

com.cisco.jtapi.extensions.CiscoConsultCallActiveEv		
public static final int	ID	1073754116

CiscoFeatureReason

com.cisco.jtapi.extensions.CiscoFeatureReason		
public static final int	REASON_BARGE	20
public static final int	REASON_BLINDTRANSFER	7

com.cisco.jtapi.extensions.CiscoFeatureReason		
public static final int	REASON_CALLPICKUP	11
public static final int	REASON_CCM_REDIRECTION	26
public static final int	REASON_CLICK_TO_CONFERENCE	39
public static final int	REASON_CONFERENCE	9
public static final int	REASON_DPARK_CALLPARK	27
public static final int	REASON_DPARK_REVERSION	28
public static final int	REASON_DPARK_UNPARK	29
public static final int	REASON_FAC_CMC	22
public static final int	REASON_FORWARDALL	5
public static final int	REASON_FORWARDBUSY	4
public static final int	REASON_FORWARDNOANSWER	3
public static final int	REASON_IMMDIVERT	21
public static final int	REASON_MOBILITY	33
public static final int	REASON_MOBILITY_CELLPICKUP	35
public static final int	REASON_MOBILITY_FOLLOWME	38
public static final int	REASON_MOBILITY_HANDIN	36
public static final int	REASON_MOBILITY_HANDOUT	37
public static final int	REASON_MOBILITY_IVR	34
public static final int	REASON_NORMAL	12
public static final int	REASON_PARK	10
public static final int	REASON_PARKREMAINDER	15
public static final int	REASON_PARKREMINDER	15
public static final int	REASON_QSIG_PR	23
public static final int	REASON_REDIRECT	6
public static final int	REASON_REFER	24
public static final int	REASON_REPLACE	25
public static final int	REASON_SILENTMONITORING	31
public static final int	REASON_TRANSFER	2
public static final int	REASON_UNPARK	16

CiscoG711MediaCapability

com.cisco.jtapi.extensions.CiscoG711MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoG723MediaCapability

com.cisco.jtapi.extensions.CiscoG723MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoG729MediaCapability

com.cisco.jtapi.extensions.CiscoG729MediaCapability		
public static final int	FRAMESIZE_SIXTY_MILLISECOND_PACKET	60
public static final int	FRAMESIZE_THIRTY_MILLISECOND_PACKET	30
public static final int	FRAMESIZE_TWENTY_MILLISECOND_PACKET	20

CiscoGSMMediaCapability

com.cisco.jtapi.extensions.CiscoGSMMediaCapability		
public static final int	FRAMESIZE_EIGHTY_MILLISECOND_PACKET	80

CiscoJtapiException

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	ASSOCIATED_LINE_NOT_OPEN	-1932787685
public static final int	CALL_ALREADY_EXISTS	-1932787705
public static final int	CALL_DROPPED	-1932787564
public static final int	CALLHANDLE_NOTINCOMINGCALL	-1932787702
public static final int	CALLHANDLE_UNKNOWN_TO_LINECONTROL	-1932787644
public static final int	CANNOT_OPEN_DEVICE	-1932787647
public static final int	CANNOT_TERMINATE_MEDIA_ON_PHONE	-1932787690
public static final int	CFWDALL_ALREADY_SET	-1932787597
public static final int	CFWDALL_DESTN_INVALID	-1932787596
public static final int	CLUSTER_LINK_FAILURE	-1932787612
public static final int	COMMAND_NOT_IMPLEMENTED_ON_DEVICE	-1932787559
public static final int	CONFERENCE_ALREADY_PRESENT	-1932787588
public static final int	CONFERENCE_FAILED	-1932787590
public static final int	CONFERENCE_FULL	-1932787642

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CONFERENCE_INACTIVE	-1932787587
public static final int	CONFERENCE_INVALID_PARTICIPANT	-1932787589
public static final int	CTIERR_ACCESS_TO_DEVICE_DENIED	-1932787688
public static final int	CTIERR_APP_SOFTKEYS_ALREADY_CONTROLLED	-1932787679
public static final int	CTIERR_APPLICATION_DATA_SIZE_EXCEEDED	-1932787675
public static final int	CTIERR_BIB_NOT_CONFIGURED	-1932787476
public static final int	CTIERR_BIB_RESOURCE_NOT_AVAILABLE	-1932787489
public static final int	CTIERR_CALL_MANAGER_NOT_AVAILABLE	-1932787689
public static final int	CTIERR_CALL_NOT_EXISTED	-1932787533
public static final int	CTIERR_CALL_PARK_NO_DN	-1932787579
public static final int	CTIERR_CALL_REQUEST_ALREADY_OUTSTANDING	-1932787577
public static final int	CTIERR_CALL_UNPARK_FAILED	-1932787583
public static final int	CTIERR_CAPABILITIES_DO_NOT_MATCH	-1932787518
public static final int	CTIERR_CLOSE_DELAY_NOT_SUPPORTED_WITH_REG_TYPE	-1932787673
public static final int	CTIERR_CONFERENCE_ALREADY_EXISTED	-1932787535
public static final int	CTIERR_CONFERENCE_NOT_EXISTED	-1932787534
public static final int	CTIERR_CONNECTION_ON_INVALID_PORT	-1932787503
public static final int	CTIERR_CONSULT_CALL_FAILURE	-1932787576
public static final int	CTIERR_CONSULTCALL_ALREADY_OUTSTANDING	-1932787640
public static final int	CTIERR_CRYPTO_CAPABILITY_MISMATCH	-1932787500
public static final int	CTIERR_CTIHANDLER_PROCESS_CREATION_FAILED	-1932787515
public static final int	CTIERR_DB_INITIALIZATION_ERROR	-1932787494
public static final int	CTIERR_DEVICE_ALREADY_OPENED	-1932787552
public static final int	CTIERR_DEVICE_NOT_OPENED_YET	-1932787551
public static final int	CTIERR_DEVICE_OWNER_ALIVE_TIMER_STARTED	-1932787517
public static final int	CTIERR_DEVICE_REGISTRATION_FAILED_NOT_SUPPORTED_MEDIATYPE	-1932787490
public static final int	CTIERR_DEVICE_RESTRICTED	-1932787502
public static final int	CTIERR_DEVICE_SHUTTING_DOWN	-1932787558
public static final int	CTIERR_DIRECTORY_LOGIN_TIMEOUT	-1932787595
public static final int	CTIERR_DUPLICATE_CALL_REFERENCE	-1932787529
public static final int	CTIERR_DYNREG_IPADDRMODE_MISMATCH	-1932787468
public static final int	CTIERR_FAC_CMC_REASON_CMC_INVALID	-1932787506
public static final int	CTIERR_FAC_CMC_REASON_CMC_NEEDED	-1932787509

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	-1932787508
public static final int	CTIERR_FAC_CMC_REASON_FAC_INVALID	-1932787507
public static final int	CTIERR_FAC_CMC_REASON_FAC_NEEDED	-1932787510
public static final int	CTIERR_FEATURE_ALREADY_REGISTERED	-1932787575
public static final int	CTIERR_FEATURE_DATA_REJECT	-1932787565
public static final int	CTIERR_FEATURE_SELECT_FAILED	-1932787514
public static final int	CTIERR_ILLEGAL_DEVICE_TYPE	-1932787578
public static final int	CTIERR_INCOMPATIBLE_AUTOINSTALL_PROTOCOL_VERSION	-1932787629
public static final int	CTIERR_INCORRECT_MEDIA_CAPABILITY	-1932787560
public static final int	CTIERR_INFORMATION_NOT_AVAILABLE	-1932787677
public static final int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_CONFIGURED	-1932787475
public static final int	CTIERR_INTERCOM_SPEEDDIAL_ALREADY_SET	-1932787493
public static final int	CTIERR_INTERCOM_SPEEDDIAL_DESTN_INVALID	-1932787492
public static final int	CTIERR_INTERCOM_TALKBACK_ALREADY_PENDING	-1932787474
public static final int	CTIERR_INTERCOM_TALKBACK_FAILURE	-1932787491
public static final int	CTIERR_INTERNAL_FAILURE	-1932787568
public static final int	CTIERR_INVALID_CALLID	-1932787487
public static final int	CTIERR_INVALID_DEVICE_NAME	-1932787678
public static final int	CTIERR_INVALID_DTMFDIGITS	-1932787561
public static final int	CTIERR_INVALID_FILTER_SIZE	-1932787625
public static final int	CTIERR_INVALID_MEDIA_DEVICE	-1932787674
public static final int	CTIERR_INVALID_MEDIA_PARAMETER	-1932787554
public static final int	CTIERR_INVALID_MEDIA_PROCESS	-1932787519
public static final int	CTIERR_INVALID_MEDIA_RESOURCE_ID	-1932787557
public static final int	CTIERR_INVALID_MESSAGE_HEADER_INFO	-1932787627
public static final int	CTIERR_INVALID_MESSAGE_LENGTH	-1932787628
public static final int	CTIERR_INVALID_MONITOR_DESTN	-1932787486
public static final int	CTIERR_INVALID_MONITOR_DN_TYPE	-1932787580
public static final int	CTIERR_INVALID_MONITORMODE	-1932787473
public static final int	CTIERR_INVALID_PARAMETER	-1932787532
public static final int	CTIERR_INVALID_PARK_DN	-1932787582
public static final int	CTIERR_INVALID_PARK_REGISTRATION_HANDLE	-1932787581
public static final int	CTIERR_INVALID_RESOURCE_TYPE	-1932787530
public static final int	CTIERR_IPADDRMODE_MISMATCH	-1932787469

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_LINE_OUT_OF_SERVICE	-1932787594
public static final int	CTIERR_LINE_RESTRICTED	-1932787501
public static final int	CTIERR_MAXCALL_LIMIT_REACHED	-1932787516
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_DYNAMIC	-1932787548
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_NONE	-1932787550
public static final int	CTIERR_MEDIA_ALREADY_TERMINATED_STATIC	-1932787549
public static final int	CTIERR_MEDIA_CAPABILITY_MISMATCH	-1932787553
public static final int	CTIERR_MEDIA_RESOURCE_NAME_SIZE_EXCEEDED	-1932787676
public static final int	CTIERR_MEDIAREGISTRATIONTYPE_DO_NOT_MATCH	-1932787567
public static final int	CTIERR_MESSAGE_TOO_BIG	-1932787626
public static final int	CTIERR_MORE_ACTIVE_CALLS_THAN_RESERVED	-1932787531
public static final int	CTIERR_NO_EXISTING_CALLS	-1932787512
public static final int	CTIERR_NO_EXISTING_CONFERENCE	-1932787527
public static final int	CTIERR_NO_RECORDING_SESSION	-1932787479
public static final int	CTIERR_NO_RESPONSE_FROM_MP	-1932787526
public static final int	CTIERR_NOT_PRESERVED_CALL	-1932787528
public static final int	CTIERR_OPERATION_FAILED_QUIETCLEAR	-1932787566
public static final int	CTIERR_OPERATION_NOT_ALLOWED	-1932787555
public static final int	CTIERR_OUT_OF_BANDWIDTH	-1932787498
public static final int	CTIERR_OWNER_NOT_ALIVE	-1932787547
public static final int	CTIERR_PENDING_ACCEPT_OR_ANSWER_REQUEST	-1932787520
public static final int	CTIERR_PENDING_START_MONITORING_REQUEST	-1932787485
public static final int	CTIERR_PENDING_START_RECORDING_REQUEST	-1932787483
public static final int	CTIERR_PENDING_STOP_RECORDING_REQUEST	-1932787482
public static final int	CTIERR_PRIMARY_CALL_INVALID	-1932787471
public static final int	CTIERR_PRIMARY_CALL_STATE_INVALID	-1932787470
public static final int	CTIERR_RECORDING_ALREADY_INPROGRESS	-1932787480
public static final int	CTIERR_RECORDING_CONFIG_NOT_MATCHING	-1932787477
public static final int	CTIERR_RECORDING_SESSION_INACTIVE	-1932787478
public static final int	CTIERR_REDIRECT_UNAUTHORIZED_COMMAND_USAGE	-1932787513

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	CTIERR_REGISTER_FEATURE_ACTIVATION_FAILED	-1932787585
public static final int	CTIERR_REGISTER_FEATURE_APP_ALREADY_REGISTERED	-1932787523
public static final int	CTIERR_REGISTER_FEATURE_PROVIDER_NOT_REGISTERED	-1932787524
public static final int	CTIERR_RESOURCE_NOT_AVAILABLE	-1932787536
public static final int	CTIERR_START_MONITORING_FAILED	-1932787484
public static final int	CTIERR_START_RECORDING_FAILED	-1932787481
public static final int	CTIERR_STATION_SHUT_DOWN	-1932787574
public static final int	CTIERR_SYSTEM_ERROR	-1932787525
public static final int	CTIERR_UDP_PASS_THROUGH_NOT_SUPPORTED	-1932787638
public static final int	CTIERR_UNKNOWN_EXCEPTION	-1932787556
public static final int	CTIERR_UNSUPPORTED_CALL_PARK_TYPE	-1932787584
public static final int	CTIERR_UNSUPPORTED_CFWD_TYPE	-1932787511
public static final int	CTIERR_USER_NOT_AUTH_FOR_SECURITY	-1932787504
public static final int	DARES_INVALID_REQ_TYPE	-1932787591
public static final int	DATA_SIZE_LIMIT_EXCEEDED	-1932787681
public static final int	DB_ERROR	-1932787691
public static final int	DB_ILLEGAL_DEVICE_TYPE	-1932787692
public static final int	DB_NO_MORE_DEVICES	-1932787694
public static final int	DESTINATION_BUSY	-1897005054
public static final int	DESTINATION_UNKNOWN	-1897005055
public static final int	DEVICE_ALREADY_REGISTERED	-1932787693
public static final int	DEVICE_NOT_OPEN	-1932787686
public static final int	DEVICE_OUT_OF_SERVICE	-1932787593
public static final int	DIGIT_GENERATION_ALREADY_IN_PROGRESS	-1932787610
public static final int	DIGIT_GENERATION_CALLSTATE_CHANGED	-1932787607
public static final int	DIGIT_GENERATION_WRONG_CALL_HANDLE	-1932787609
public static final int	DIGIT_GENERATION_WRONG_CALL_STATE	-1932787608
public static final int	DIRECTORY_LOGIN_FAILED	-1932787616
public static final int	DIRECTORY_LOGIN_NOT_ALLOWED	-1932787617
public static final int	DIRECTORY_TEMPORARY_UNAVAILABLE	-1932787618
public static final int	EXISTING_FIRSTPARTY	-1932787709
public static final int	HOLDFAILED	-1932787697
public static final int	ILLEGAL_CALLINGPARTY	-1932787706
public static final int	ILLEGAL_CALLSTATE	-1932787703
public static final int	ILLEGAL_HANDLE	-1932787708
public static final int	ILLEGAL_MESSAGE_FORMAT	-1932787630

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	INCOMPATIBLE_PROTOCOL_VERSION	-1932787632
public static final int	INVALID_LINE_HANDLE	-1932787599
public static final int	INVALID_RING_OPTION	-1932787680
public static final int	LINE_GREATER_THAN_MAX_LINE	-1932787606
public static final int	LINE_INFO_DOES_NOT_EXIST	-1932787611
public static final int	LINE_NOT_PRIMARY	-1932787598
public static final int	LINECONTROL_FAILURE	-1932787704
public static final int	MAX_NUMBER_OF_CTI_CONNECTIONS_REACHED	-1932787641
public static final int	MSGWAITING_DESTN_INVALID	-1932787592
public static final int	NO_ACTIVE_DEVICE_FOR_THIRDPARTY	-1932787710
public static final int	NO_CONFERENCE_BRIDGE	-1932787639
public static final int	NOT_INITIALIZED	-1932787613
public static final int	PROTOCOL_TIMEOUT	-1091584273
public static final int	PROVIDER_ALREADY_OPEN	-1932787614
public static final int	PROVIDER_CLOSED	-559038737
public static final int	PROVIDER_NOT_OPEN	-1932787615
public static final int	REDIRECT_CALL_CALL_TABLE_FULL	-1932787662
public static final int	REDIRECT_CALL_DESTINATION_BUSY	-1932787649
public static final int	REDIRECT_CALL_DESTINATION_OUT_OF_ORDER	-1932787648
public static final int	REDIRECT_CALL_DIGIT_ANALYSIS_TIMEOUT	-1932787659
public static final int	REDIRECT_CALL_DOES_NOT_EXIST	-1932787683
public static final int	REDIRECT_CALL_INCOMPATIBLE_STATE	-1932787654
public static final int	REDIRECT_CALL_MEDIA_CONNECTION_FAILED	-1932787658
public static final int	REDIRECT_CALL_NORMAL_CLEARING	-1932787651
public static final int	REDIRECT_CALL_ORIGINATOR_ABANDONED	-1932787656
public static final int	REDIRECT_CALL_PARTY_TABLE_FULL	-1932787657
public static final int	REDIRECT_CALL_PENDING_REDIRECT_TRANSACTION	-1932787653
public static final int	REDIRECT_CALL_PROTOCOL_ERROR	-1932787661
public static final int	REDIRECT_CALL_UNKNOWN_DESTINATION	-1932787660
public static final int	REDIRECT_CALL_UNKNOWN_ERROR	-1932787652
public static final int	REDIRECT_CALL_UNKNOWN_PARTY	-1932787655
public static final int	REDIRECT_CALL_UNRECOGNIZED_MANAGER	-1932787650
public static final int	REDIRECT_CALLINFO_ERR	-1932787664
public static final int	REDIRECT_ERR	-1932787663
public static final int	RETRIEVEFAILED	-1932787695
public static final int	RETRIEVEFAILED_ACTIVE_CALL_ON_LINE	-1932787600

com.cisco.jtapi.extensions.CiscoJtapiException		
public static final int	SSAPI_NOT_REGISTERED	-1932787684
public static final int	TIMEOUT	-1932787711
public static final int	TRANSFER_INACTIVE	-1932787586
public static final int	TRANSFERFAILED	-1932787698
public static final int	TRANSFERFAILED_CALLCONTROL_TIMEOUT	-1932787645
public static final int	TRANSFERFAILED_DESTINATION_BUSY	-1932787699
public static final int	TRANSFERFAILED_DESTINATION_UNALLOCATED	-1932787701
public static final int	TRANSFERFAILED_OUTSTANDING_TRANSFER	-1932787646
public static final int	UNDEFINED_LINE	-1932787707
public static final int	UNKNOWN_GLOBAL_CALL_HANDLE	-1932787687
public static final int	UNRECOGNIZABLE_PDU	-1932787631
public static final int	UNSPECIFIED	0

CiscoLocales

com.cisco.jtapi.extensions.CiscoLocales		
public static final int	LOCALE_ARABIC_ALGERIA	47
public static final int	LOCALE_ARABIC_BAHRAIN	48
public static final int	LOCALE_ARABIC_EGYPT	49
public static final int	LOCALE_ARABIC_IRAQ	50
public static final int	LOCALE_ARABIC_JORDAN	51
public static final int	LOCALE_ARABIC_KUWAIT	38
public static final int	LOCALE_ARABIC_LEBANON	52
public static final int	LOCALE_ARABIC_MOROCCO	53
public static final int	LOCALE_ARABIC_OMAN	36
public static final int	LOCALE_ARABIC_QATAR	54
public static final int	LOCALE_ARABIC_SAUDI_ARABIA	37
public static final int	LOCALE_ARABIC_TUNISIA	55
public static final int	LOCALE_ARABIC_UNITED_ARAB_EMIRATES	35
public static final int	LOCALE_ARABIC_YEMEN	56
public static final int	LOCALE_BULGARIAN_BULGARIA	27
public static final int	LOCALE_CATALAN_SPAIN	32
public static final int	LOCALE_CHINESE_HONG_KONG	24
public static final int	LOCALE_CROATIAN_CROATIA	28
public static final int	LOCALE_CZECH_CZECH_REPUBLIC	26
public static final int	LOCALE_DANISH_DENMARK	12
public static final int	LOCALE_DUTCH_NETHERLAND	8

com.cisco.jtapi.extensions.CiscoLocales		
public static final int	LOCALE_ENGLISH_UNITED_KINGDOM	33
public static final int	LOCALE_ENGLISH_UNITED_STATES	1
public static final int	LOCALE_FINNISH_FINLAND	22
public static final int	LOCALE_FRENCH_FRANCE	2
public static final int	LOCALE_GERMAN_GERMANY	3
public static final int	LOCALE_GREEK_GREECE	16
public static final int	LOCALE_HEBREW_ISRAEL	39
public static final int	LOCALE_HUNGARIAN_HUNGARY	14
public static final int	LOCALE_ITALIAN_ITALY	7
public static final int	LOCALE_JAPANESE_JAPAN	13
public static final int	LOCALE_KOREAN_KOREA	21
public static final int	LOCALE_NORWEGIAN_NORWAY	9
public static final int	LOCALE_POLISH_POLAND	15
public static final int	LOCALE_PORTUGUESE_BRAZIL	23
public static final int	LOCALE_PORTUGUESE_PORTUGAL	10
public static final int	LOCALE_ROMANIAN_ROMANIA	30
public static final int	LOCALE_RUSSIAN_RUSSIA	5
public static final int	LOCALE_SERBIAN_REPUBLIC_OF_MONTENEGRO	41
public static final int	LOCALE_SERBIAN_REPUBLIC_OF_SERBIA	40
public static final int	LOCALE_SIMPLIFIED_CHINESE_CHINA	20
public static final int	LOCALE_SLOVAK_SLOVAKIA	25
public static final int	LOCALE_SLOVENIAN_SLOVENIA	29
public static final int	LOCALE_SPANISH_SPAIN	6
public static final int	LOCALE_SWEDISH_SWEDEN	11
public static final int	LOCALE_THAI_THAILAND	42
public static final int	LOCALE_TRADITIONAL_CHINESE_CHINA	19

CiscoMediaConnectionMode

com.cisco.jtapi.extensions.CiscoMediaConnectionMode		
public static final int	NONE	0
public static final int	RECEIVE_ONLY	1
public static final int	TRANSMIT_AND_RECEIVE	3
public static final int	TRANSMIT_ONLY	2

CiscoMediaEncryptionAlgorithmType

com.cisco.jtapi.extensions.CiscoMediaEncryptionAlgorithmType		
public static final int	AES_128_COUNTER	1

CiscoMediaOpenLogicalChannelEv

com.cisco.jtapi.extensions.CiscoMediaOpenLogicalChannelEv		
public static final int	ID	1073758213

CiscoMediaSecurityIndicator

com.cisco.jtapi.extensions.CiscoMediaSecurityIndicator		
public static final int	MEDIA_ENCRYPT_KEYS_AVAILABLE	0
public static final int	MEDIA_ENCRYPT_KEYS_UNAVAILABLE	2
public static final int	MEDIA_ENCRYPT_USER_NOT_AUTHORIZED	1
public static final int	MEDIA_NOT_ENCRYPTED	3

CiscoOutOfServiceEv

com.cisco.jtapi.extensions.CiscoOutOfServiceEv		
public static final int	CAUSE_CALLMANAGER_FAILURE	1001
public static final int	CAUSE_CTIMANAGER_FAILURE	1007
public static final int	CAUSE_DEVICE_FAILURE	1004
public static final int	CAUSE_DEVICE_RESTRICTED	1008
public static final int	CAUSE_DEVICE_UNREGISTERED	1005
public static final int	CAUSE_LINE_RESTRICTED	1009
public static final int	CAUSE_NOCALLMANAGER_AVAILABLE	1003
public static final int	CAUSE_REHOME_TO_HIGHER_PRIORITY_CM	1002
public static final int	CAUSE_REHOMING_FAILURE	1006
public static final int	ID	1073750021

CiscoPartyInfo

com.cisco.jtapi.extensions.CiscoPartyInfo		
public static final int	ABBREVIATED_NUMBER	6
public static final int	INTERNATIONAL_NUMBER	1
public static final int	NATIONAL_NUMBER	2
public static final int	NET_SPECIFIC_NUMBER	3
public static final int	RESERVED_FOR_EXTENSION	7
public static final int	SUBSCRIBER_NUMBER	4
public static final int	UNKNOWN_NUMBER	0

CiscoProvCallParkEv

com.cisco.jtapi.extensions.CiscoProvCallParkEv		
public static final int	ID	1073754113
public static final int	PARK_STATE_ACTIVE	2
public static final int	PARK_STATE_IDLE	1
public static final int	REASON_CALLPARK	1
public static final int	REASON_CALLPARKREMAINDER	3
public static final int	REASON_CALLPARKREMINDER	3
public static final int	REASON_CALLUNPARK	2

CiscoProvFeatureID

com.cisco.jtapi.extensions.CiscoProvFeatureID		
public static final int	MONITOR_CALLPARK_DN	1234

CiscoProviderCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv		
public static final int	ID	1073741846
public static final int	MODIFY_CGPN	32
public static final int	MONITOR_PARKDN	64
public static final int	SUPERPROVIDER	16

CiscoProvTerminalCapabilityChangedEv

com.cisco.jtapi.extensions.CiscoProvTerminalCapabilityChangedEv		
public static final int	ID	1073741847

CiscoRestrictedEv

com.cisco.jtapi.extensions.CiscoRestrictedEv		
public static final int	CAUSE_UNKNOWN	0
public static final int	CAUSE_UNSUPPORTED_DEVICE_CONFIGURATION	3
public static final int	CAUSE_UNSUPPORTED_PROTOCOL	2
public static final int	CAUSE_USER_RESTRICTED	1
public static final int	ID	1073741833

CiscoRouteSession

com.cisco.jtapi.extensions.CiscoRouteSession		
public static final int		
public static final int	CALLINGADDRESS_SEARCH_SPACE	1
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_INVALID	-1932787506
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_CMC_NEEDED	-1932787509
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_CMC_NEEDED	-1932787508
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_INVALID	-1932787507
public static final int	CAUSE_CTIERR_FAC_CMC_REASON_FAC_NEEDED	-1932787510
public static final int	DEFAULT_SEARCH_SPACE	0
public static final int	DONOT_RESET_ORIGINALCALLED	0
public static final int	ERROR_INVALID_STATE	7
public static final int	ERROR_NO_CALLBACK	6
public static final int	ERROR_NONE	4
public static final int	ERROR_ROUTESELECT_TIMEOUT	5
public static final int	RESET_ORIGINALCALLED	1
public static final int	ROUTEADDRESS_SEARCH_SPACE	2

CiscoRouteTerminal

com.cisco.jtapi.extensions.CiscoRouteTerminal		
public static final int	DYNAMIC_MEDIA_REGISTRATION	2
public static final int	NO_MEDIA_REGISTRATION	0

CiscoRTPBitRate

com.cisco.jtapi.extensions.CiscoRTPBitRate		
public static final int	R5_3	1
public static final int	R6_4	2

CiscoRTPInputKeyEv

com.cisco.jtapi.extensions.CiscoRTPInputKeyEv		
public static final int	ID	1073758214

CiscoRTPInputStartedEv

com.cisco.jtapi.extensions.CiscoRTPInputStartedEv		
public static final int	ID	1073758209

CiscoRTPInputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPInputStoppedEv		
public static final int	ID	1073758210

CiscoRTPOutputKeyEv

com.cisco.jtapi.extensions.CiscoRTPOutputKeyEv		
public static final int	ID	1073758215

CiscoRTPOutputStartedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv		
public static final int	ID	1073758211

CiscoRTPOutputStoppedEv

com.cisco.jtapi.extensions.CiscoRTPOutputStartedEv		
public static final int	ID	1073758212

CiscoRTPPayload

com.cisco.jtapi.extensions.CiscoRTPPayload		
public static final int	ACTIVEVOICE	81
public static final int	ACY_G729AASSN	15
public static final int	DATA56	33
public static final int	DATA64	32
public static final int	G711ALAW56K	3
public static final int	G711ALAW64K	2
public static final int	G711ULAW56K	5
public static final int	G711ULAW64K	4
public static final int	G722_48K	8
public static final int	G722_56K	7
public static final int	G722_64K	6
public static final int	G7231	9
public static final int	G728	10
public static final int	G729	11
public static final int	G729ANNEXA	12
public static final int	GSM	80
public static final int	IS11172AUDIOCAP	13
public static final int	IS13818AUDIOCAP	14
public static final int	NONSTANDARD	1
public static final int	WIDEBAND_256K	25

CiscoTermActivatedEv

com.cisco.jtapi.extensions.CiscoTermActivatedEv		
public static final int	ID	1073741843

CiscoTermButtonPressedEv

com.cisco.jtapi.extensions.CiscoTermButtonPressedEv		
public static final int	CHARA	10
public static final int	CHARB	11
public static final int	CHARC	12
public static final int	CHARD	13
public static final int	EIGHT	8
public static final int	FIVE	5
public static final int	FOUR	4
public static final int	ID	1073745936
public static final int	NINE	9
public static final int	ONE	1
public static final int	POUND	15
public static final int	SEVEN	7
public static final int	SIX	6
public static final int	STAR	14
public static final int	THREE	3
public static final int	TWO	2
public static final int	ZERO	0

CiscoTermConnMonitoringEndEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringEndEv		
public static final int	ID	1073762311

CiscoTermConnMonitoringStartEv

com.cisco.jtapi.extensions.CiscoTermConnMonitoringStartEv		
public static final int	ID	1073762310

CiscoTermConnMonitorInitiatorInfoEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorInitiatorInfoEv		
public static final int	ID	1073762313

CiscoTermConnMonitorTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnMonitorTargetInfoEv		
public static final int	ID	1073762314

CiscoTermConnPrivacyChangedEv

com.cisco.jtapi.extensions.CiscoTermConnPrivacyChangedEv		
public static final int	ID	1073762305

CiscoTermConnRecordingEndEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingEndEv		
public static final int	ID	1073762309

CiscoTermConnRecordingStartEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingStartEv		
public static final int	ID	1073762308

CiscoTermConnRecordingTargetInfoEv

com.cisco.jtapi.extensions.CiscoTermConnRecordingTargetInfoEv		
public static final int	ID	1073762312

CiscoTermConnSelectChangedEv

com.cisco.jtapi.extensions.CiscoTermConnSelectChangedEv		
public static final int	ID	1073762307

CiscoTermCreatedEv

com.cisco.jtapi.extensions.CiscoTermCreatedEv		
public static final int	ID	1073745921

CiscoTermDataEv

com.cisco.jtapi.extensions.CiscoTermDataEv		
public static final int	ID	1073745922

CiscoTermDeviceStateActiveEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateActiveEv		
public static final int	ID	1073745926

CiscoTermDeviceStateAlertingEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateAlertingEv		
public static final int	ID	1073745927

CiscoTermDeviceStateHeldEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateHeldEv		
public static final int	ID	1073745928

CiscoTermDeviceStateIdleEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateIdleEv		
public static final int	ID	1073745929

CiscoTermDeviceStateWhisperEv

com.cisco.jtapi.extensions.CiscoTermDeviceStateWhisperEv		
public static final int	ID	1073745941

CiscoTermDNDOptionChangedEv

com.cisco.jtapi.extensions.CiscoTermDNDOptionChangedEv		
public static final int	ID	1073745942

CiscoTermDNDStatusChangedEv

com.cisco.jtapi.extensions.CiscoTermDNDStatusChangedEv		
public static final int	ID	1073745940

CiscoTerminal

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	ASCII_ENCODING	2
public static final int	DEVICESTATE_ACTIVE	1
public static final int	DEVICESTATE_ALERTING	2
public static final int	DEVICESTATE_HELD	3
public static final int	DEVICESTATE_IDLE	0
public static final int	DEVICESTATE_UNKNOWN	4
public static final int	DEVICESTATE_WHISPER	5
public static final int	DND_OPTION_CALL_REJECT	2
public static final int	DND_OPTION_NONE	0
public static final int	DND_OPTION_RINGER_OFF	1
public static final int	IN_SERVICE	1
public static final int	IP_ADDRESSING_MODE_IPV4	0

com.cisco.jtapi.extensions.CiscoTerminal		
public static final int	IP_ADDRESSING_MODE_IPV4_V6	2
public static final int	IP_ADDRESSING_MODE_IPV6	1
public static final int	IP_ADDRESSING_MODE_UNKNOWN	3
public static final int	IP_ADDRESSING_MODE_UNKNOWN_ANATRED	4
public static final int	NOT_APPLICABLE	1
public static final int	OUT_OF_SERVICE	0
public static final int	UCS2UNICODE_ENCODING	3
public static final int	UNKNOWN_ENCODING	0

CiscoTerminalConnection

com.cisco.jtapi.extensions.CiscoTerminalConnection		
public static final int	CISCO_SELECTEDLOCAL	1
public static final int	CISCO_SELECTEDNONE	0
public static final int	CISCO_SELECTEDREMOTE	2

CiscoTerminalProtocol

com.cisco.jtapi.extensions.CiscoTerminalProtocol		
public static final int	PROTOCOL_NONE	0
public static final int	PROTOCOL_SCCP	1
public static final int	PROTOCOL_SIP	2

CiscoTermInServiceEv

com.cisco.jtapi.extensions.CiscoTermInServiceEv		
public static final int	ID	1073745923

CiscoTermOutOfServiceEv

com.cisco.jtapi.extensions.CiscoTermOutOfServiceEv		
public static final int	ID	1073745924

CiscoTermRegistrationFailedEv

com.cisco.jtapi.extensions.CiscoTermRegistrationFailedEv		
public static final int	DB_INITIALIZATION_ERROR	15
public static final int	ID	1073745937
public static final int	IP_ADDRESSING_MODE_MISMATCH	19
public static final int	MEDIA_ALREADY_TERMINATED_DYNAMIC	8
public static final int	MEDIA_ALREADY_TERMINATED_NONE	6
public static final int	MEDIA_ALREADY_TERMINATED_STATIC	7
public static final int	MEDIA_CAPABILITY_MISMATCH	10
public static final int	OWNER_NOT_ALIVE	11
public static final int	UNKNOWN	0

CiscoTermRemovedEv

com.cisco.jtapi.extensions.CiscoTermRemovedEv		
public static final int	ID	1073745925

CiscoTermRestrictedEv

com.cisco.jtapi.extensions.CiscoTermRestrictedEv		
public static final int	ID	1073741841

CiscoTermSnapshotCompletedEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv		
public static final int	ID	1073745939

CiscoTermSnapshotEv

com.cisco.jtapi.extensions.CiscoTermSnapshotCompletedEv		
public static final int	ID	1073745938

CiscoTone

com.cisco.jtapi.extensions.CiscoTone		
public static final int	ZIPZIP	49

CiscoToneChangedEv

com.cisco.jtapi.extensions.CiscoToneChangedEv		
public static final int	CMC_REQUIRED	1
public static final int	FAC_CMC_REQUIRED	2
public static final int	FAC_REQUIRED	0
public static final int	ID	1073754119

CiscoTransferEndEv

com.cisco.jtapi.extensions.CiscoTransferEndEv		
public static final int	ID	1073754117

CiscoTransferStartEv

com.cisco.jtapi.extensions.CiscoTransferStartEv		
public static final int	ID	1073754118

CiscoUrlInfo

com.cisco.jtapi.extensions.CiscoUrlInfo		
public static final int	TRANSPORT_TYPE_TCP	1
public static final int	TRANSPORT_TYPE_UDP	2
public static final int	URL_TYPE_SIP	1
public static final int	URL_TYPE_TEL	2
public static final int	URL_TYPE_UNKNOWN	0

CiscoWideBandMediaCapability

com.cisco.jtapi.extensions.CiscoWideBandMediaCapability		
public static final int	FRAMESIZE_TEN_MILLISECOND_PACKET	10

Alarm

com.cisco.services.alarm.Alarm		
public static final int	ALERTS	1
public static final int	CRITICAL	2
public static final int	DEBUGGING	7
public static final int	EMERGENCIES	0
public static final int	ERROR	3
public static final int	HIGHEST_LEVEL	7
public static final int	INFORMATIONAL	6
public static final int	LOWEST_LEVEL	0
public static final int	NO_SEVERITY	-1
public static final int	NOTIFICATION	5
public static final java.lang.String	UNKNOWN_MNEMONIC	"UNK"
public static final int	WARNING	4

LogFileTraceWriter

com.cisco.services.tracing.LogFileTraceWriter		
public static final java.lang.String	DEFAULT_FILE_NAME_BASE	"trace"
public static final java.lang.String	DEFAULT_FILE_NAME_EXTENSION	"log"
public static final char	DIR_BASE_NAME_NUM_SEPERATOR	95
public static final int	MIN_FILE_SIZE	10240
public static final int	MIN_FILES	2
public static final int	ROLLOVER_THRESHOLD	1024

Trace

com.cisco.services.tracing.Trace		
public static final int	ALERTS	1
public static final java.lang.String	ALERTS_TRACE_NAME	"ALERTS"
public static final int	CRITICAL	2
public static final java.lang.String	CRITICAL_TRACE_NAME	"CRITICAL"
public static final int	DEBUGGING	7
public static final java.lang.String	DEBUGGING_TRACE_NAME	"DEBUGGING"
public static final int	EMERGENCIES	0
public static final java.lang.String	EMERGENCIES_TRACE_NAME	"EMERGENCIES"
public static final int	ERROR	3
public static final java.lang.String	ERROR_TRACE_NAME	"ERROR"
public static final int	HIGHEST_LEVEL	7
public static final int	INFORMATIONAL	6
public static final java.lang.String	INFORMATIONAL_TRACE_NAME	"INFORMATIONAL"
public static final int	LOWEST_LEVEL	0
public static final int	NOTIFICATION	5
public static final java.lang.String	NOTIFICATION_TRACE_NAME	"NOTIFICATION"
public static final int	WARNING	4
public static final java.lang.String	WARNING_TRACE_NAME	"WARNING"

com.cisco.*

APPENDIX G

非推奨の API

この付録では、非推奨の API、フィールド、およびメソッドを示します。非推奨の API を使用しないことをお勧めします。これは、一般的にはその API が改善されたためであり、通常は代わりの API が提供されています。非推奨の API は、将来の実装で削除される場合があります。

この付録は次のセクションで構成されています。

非推奨インターフェイス

非推奨インターフェイス
com.cisco.jtapi.extensions.CiscoRouteAddress このインターフェイスは実装されていません。

非推奨フィールド

非推奨フィールド
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.MODIFY_CGPN この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.MONITOR_PARKDN この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。
com.cisco.jtapi.extensions.CiscoProvCallParkEv.REASON_CALLPARKREMAINDER このインターフェイスはスペル ミスのために非推奨になりました。新しいインターフェイス REASON_CALLPARKREMINDER を使用してください。

非推奨フィールド
com.cisco.jtapi.extensions.CiscoFeatureReason.REASON_PARKREMAINDER REASON_PARKREMINDER を使用してください。
com.cisco.jtapi.extensions.CiscoProviderCapabilityChangedEv.SUPERPROVIDER この定数はどのインターフェイスからも返されません。アプリケーションでは使用しないでください。

非推奨メソッド

非推奨メソッド
com.cisco.jtapi.extensions.CiscoTermDataEv.getData() byte[] getTermData を使用してください。
com.cisco.jtapi.extensions.CiscoJtapiException.getErrorDescription(int) 代わりに、String getErrorDescription (); を使用してください。
com.cisco.jtapi.extensions.CiscoJtapiException.getErrorName(int) 代わりに、String getErrorName (); を使用してください。
com.cisco.jtapi.extensions.CiscoConsultCallActiveEv.getHeldTerminalConnection() CiscoConsultCall.getConsultingTerminalConnection() に置き換えられました。
com.cisco.jtapi.extensions.CiscoCall.getLastRedirectingPartyInfo() getLastRedirectedPartyInfo() を使用してください。
com.cisco.jtapi.extensions.CiscoAddress.getRegistrationState() このメソッドは getState() メソッドに置き換えられました。
com.cisco.jtapi.extensions.CiscoTerminal.getRegistrationState() このメソッドは getState() メソッドに置き換えられました。
com.cisco.jtapi.extensions.CiscoMediaTerminal.register(InetAddress, int)
com.cisco.jtapi.extensions.CiscoTerminal.sendData(String)
com.cisco.jtapi.extensions.CiscoJtapiProperties.setSecurityPropertyForInstance(String, String, String, String, String, String, String, String, boolean) このメソッドは、オーバーロードされたメソッド setSecurityPropertyForInstance に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。
com.cisco.services.tracing.TraceManager.setSubFacilities(String[]) TraceManager.addSubFacilities メソッドに置き換えられました。
com.cisco.services.tracing.implementation.TraceManagerImpl.setSubFacilities(String[]) addSubFacilties(String[]) に置き換えられました。
com.cisco.services.tracing.TraceManager.setSubFacility(String) TraceManager.addSubFacility メソッドに置き換えられました。
com.cisco.services.tracing.implementation.TraceManagerImpl.setSubFacility(String) addSubFacility(String) に置き換えられました。

非推奨メソッド
com.cisco.jtapi.extensions.CiscoJtapiProperties.updateCertificate(String, String, String, String, String, String, String, String) このメソッドは、オーバーロードされたメソッド updateCertifcate に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。
com.cisco.jtapi.extensions.CiscoJtapiProperties.updateServerCertificate(String, String, String, String, String) このメソッドは、オーバーロードされたメソッド updateServerCertifcate に置き換えられました。このメソッドは、Java キー ストアのパスフレーズである追加のパラメータ certStorePassphrase を取ります。このメソッドにはセキュリティの脆弱性がある可能性があります。

INDEX

A

C

D

G

I

J

M

O

P

R

S

T

X

あ

い

か

き

け

こ

さ

し

せ

た

て

と

な

は

ほ

み

め

も

ゆ

よ

り

る

ろ

わ