Cisco Unified TAPI Developers Guide for Cisco Unified Communications Manager Release 10.0(1)

First Published: 2013-12-03

Last Modified: 2018-02-13

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Preface

This chapter describes the purpose, intended audience, and organization of this document and describes the conventions that convey instructions and other information. It contains the following topics:

Purpose

This document describes the Cisco Unified TAPI implementation by detailing the functions that comprise the implementation software and illustrating how to use these functions to create applications that support the Cisco Unified Communications hardware, software, and processes. You should use this document with the Cisco Unified Communications Manager manuals to develop applications.

Audience

Cisco intends this document to be for use by telephony software engineers who are developing Cisco telephony applications that require TAPI. This document assumes that the engineer is familiar with both the C or C++ languages and the Microsoft TAPI specification.

This document assumes that you have knowledge of C or C++ languages and the Microsoft TAPI specification.You must also have knowledge or experience in the following areas:

- Extensible Markup Language (XML)
- Hypertext Markup Language (HTML)
- Hypertext Transport Protocol (HTTP)
- Socket programming

- TCP/IP Protocol
- Web Service Definition Language (WSDL) 1.1
- Secure Sockets Layer (SSL)

In addition, as a user of the Cisco Unified Communications Manager APIs, you must have a firm understanding of XML Schema. For more information about XML Schema, refer to http://www.w3.org/TR/xmlschema-0/.

You must have an understanding of Cisco Unified Communications Manager and its applications. See the Related Documentation, on page xxxiii for Cisco Unified Communications Manager documents and other related technologies.

Organization

Chapter	Description
Overview, on page 1	Outlines key concepts for Cisco Unified TAPI and lists all functions that are available in the implementation.
New and Changed Information, on page 15	Provides a list new and changed features release–by–release of Cisco Unified Communications Manager (Unified CM).
Features Supported by TSP, on page 23	Describes the features that Cisco Unified TAPI Service Provider (TSP) supports
Cisco Unified TAPI Installation, on page 105	Provides installation procedures for Cisco Unified TAPI and Cisco Unified TSP.
Basic TAPI Implementation, on page 139	Describes the supported functions in the Cisco implementation of standard Microsoft TAPI v2.1.
Cisco Device-Specific Extensions, on page 313	Describes the functions that comprise the Cisco hardware-specific implementation classes.
Cisco TSP Media Driver, on page 411	Describes how Cisco Media Driver helps TAPI-based applications to provide media interaction such as play announcements, record calls, and so on.
Cisco Unified TAPI Examples, on page 437	Provides examples that illustrate the use of the Cisco Unified TAPI implementation.
Message Sequence Charts, on page 443	Lists possible call scenarios and use cases.
Cisco Unified TAPI Interfaces, on page 1069	Lists APIs that are supported or not supported.
Troubleshooting Cisco Unified TAPI, on page 1079	Describes troubleshooting techniques.
Cisco Unified TAPI Operations-by-Release, on page 1081	Lists features, line functions, messages, and structures; phone functions, messages, and structures that have been added or modified by Cisco Unified Communications Manager release.
CTI Supported Devices, on page 1091	Lists CTI supported devices.

Related Documentation

This section lists documents and URLs that provide information on Cisco Unified Communications Manager, Cisco Unified IP Phones, TAPI specifications, and the technologies that are required to develop applications.

Cisco Unified Communications Manager Release 10.0(1)—A suite of documents that relate to the installation and configuration of Cisco Unified Communications Manager. Refer to the Cisco Unified Communications Manager Documentation Guide for Release10.0(1) for a list of documents on installing and configuring Cisco Unified Communications Manager 10.0(1), including:

- Cisco Unified Communications Manager Administration Guide, Release 10.0(1)
- Cisco Unified Communications Manager System Guide, Release 10.0(1)
- Cisco Unified Communications Manager Features and Services Guide, Release 10.0(1)
- Cisco Unified Communications Manager Release Notes, Release 10.0(1)
    - Cisco Unified IP Phones and Services—A suite of documents that relate to the installation and configuration of Cisco Unified IP Phones.
    - Cisco Distributed Director—A suite of documents that relate to the installation and configuration of Cisco Distributed Director.

For more information about TAPI specifications, creating an application to use TAPI, or TAPI administration, see the following documents:

- Microsoft TAPI 2.1 Features
- http://www.microsoft.com/ntserver/techresources/commnet/tele/tapi21.asp
- Getting Started with Windows Telephony
- http://www.microsoft.com/NTServer/commserv/deployment/planguides/getstartedtele.asp
- Windows Telephony API (TAPI)
- http://www.microsoft.com/NTServer/commserv/exec/overview/tapiabout.asp
- Creating Next Generation Telephony Applications
- http://www.microsoft.com/NTServer/commserv/techdetails/prodarch/tapi21wp.asp
- The Microsoft Telephony Application Programming Interface (TAPI) Programmer's Reference
- “For the Telephony API, Press 1; For Unimodem, Press 2; or Stay on the Line”—A paper on TAPI by Hiroo Umeno, a COMM and TAPI specialist at Microsoft.

  http://www.microsoft.com/msj/0498/tapi.aspx
- “TAPI 2.1 Microsoft TAPI Client Management”
- “TAPI 2.1 Administration Tool”

Cisco Developer Network

The Cisco Developer Network (CDN) portal provides access to multiple Cisco technology developer interfaces and collaborative support communities. CDN also provides formalized support services for these interfaces to enable developers, customers, and partners to accelerate their development. The formalized process provides access to CDN Engineers who are an extension of the product technology engineering teams. CDN Engineers have access to the resources necessary to provide expert support in a timely manner.

The Cisco Developer Network Program is designed for businesses (IHV's and ISV's) interested in going to market with Cisco. The CDN Program enables members to develop compelling solutions that unify data, voice, video, and mobile communications on Cisco's powerful communications platform. The program also allows members to take advantage of Cisco's brand, market leadership position, and installed base to help drive positive business results for themselves and their customers.

For additional information about the CDN Program and CDN support services go to http://developer.cisco.com/web/devservices.

Conventions

This document uses the following conventions:

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in italics.
[ ]	Elements in square brackets are optional.
{ x | y | z }	Alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	An unquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`screen` font	Terminal sessions and information that the system displays are in **`screen`** font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.
italic screen font	Arguments for which you supply values are in italic screen font.
^	The symbol ^ represents the key labeled Control—for example, the key combination ^D in a screen display means hold down the Control key while you press the D key.
< >	Nonprinting characters, such as passwords are in angle brackets.

Notes use the following conventions:

Note Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

Tip Means the following information might help you solve a problem.

Tip Means the described action saves time. You can save time by performing the action described in the paragraph.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly What's New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation, at:

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

Subscribe to the What's New in Cisco Product Documentation as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

Cisco Product Security Overview

This product contains cryptographic features and is subject to United States and local country laws governing import, export, transfer and use. Delivery of Cisco cryptographic products does not imply third-party authority to import, export, distribute or use encryption. Importers, exporters, distributors and users are responsible for compliance with U.S. and local country laws. By using this product you agree to comply with applicable laws and regulations. If you are unable to comply with U.S. and local laws, return this product immediately.

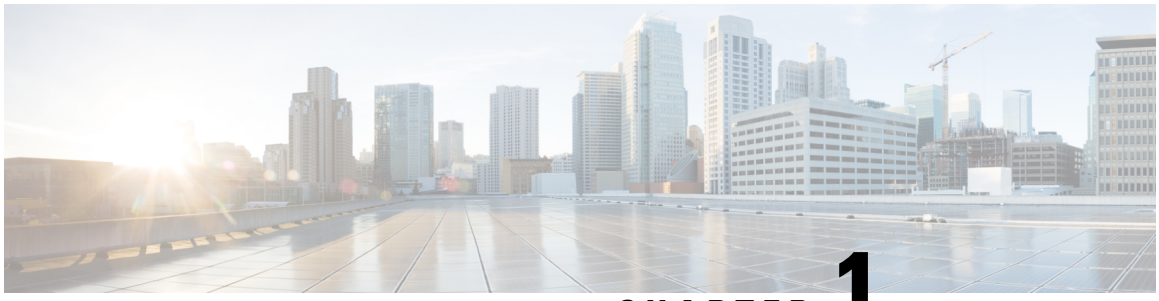
A summary of U.S. laws governing Cisco cryptographic products may be found at: http://www.cisco.com/wwl/export/crypto/tool/stqrg.html.

If you require further assistance please contact us by sending email to export@cisco.com.

OpenSSL/Open SSL Project

The following link provides information about the OpenSSL notice:

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



CHAPTER 1

Overview

Cisco Unified Communications Manager is the powerful call-processing component of the Cisco Unified Communications Solution. It is a scalable, distributable, and highly available enterprise IP telephony call-processing solution. Cisco Unified Communications Manager acts as the platform for collaborative communication and as such supports a wide array of features. In order to provision, invoke the features, monitor, and control such a powerful system, Cisco Unified Communications Manager supports different interface types.

This chapter also describes the major concepts of Cisco Unified TAPI service provider (Cisco Unified TSP) implementation. It contains the following sections:

Cisco Unified Communications Manager Interfaces

The interface types supported by Cisco Unified Communications Manager are divided into the following types:

Provisioning Interfaces

The following are the provisioning interfaces of Cisco Unified Communications Manager:

- Administration XML
- Cisco Extension Mobility service

Administrative XML

The Administration XML (AXL) API provides a mechanism for inserting, retrieving, updating and removing data from the Cisco Unified Communications Manager configuration database using an eXtensible Markup Language (XML) Simple Object Access Protocol (SOAP) interface. This allows a programmer to access Unified CM provisioning services using XML and exchange data in XML form, instead of using a binary library or DLL. The AXL methods, referred to as requests, are performed using a combination of HTTP and SOAP. SOAP is an XML remote procedure call protocol. Users perform requests by sending XML data to

the Cisco Unified Communications Manager Publisher server. The publisher then returns the AXL response, which is also a SOAP message.

For more information, See the Administrative XML Tech Center on the Cisco Developer Network http://developer.cisco.com/web/axl/home.

Cisco Extension Mobility

The Cisco Extension Mobility (Extension Mobility) service, a feature of Cisco Unified Communications Manager, allows a device, usually a Cisco Unified IP Phone, to temporarily embody a new device profile, including lines, speed dials, and services. It enables users to temporarily access their individual Cisco Unified IP Phone configuration, such as their line appearances, services, and speed dials, from other Cisco Unified IP Phones. The Extension Mobility service works by downloading a new configuration file to the phone. Cisco Unified Communications Manager dynamically generates this new configuration file based on information about the user who is logging in. You can use the XML-based Extension Mobility service API with your applications, so they can take advantage of Extension Mobility service functionality.

For more information, See the Extension Mobility API Tech Center on the Cisco Developer Network http://developer.cisco.com/web/emapi/home.

Also, see Cisco Unified Communications Manager XML Developers Guide for relevant release of Cisco Unified Communications Manager at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Device Monitoring and Call Control Interfaces

The following are the device monitoring and call control interfaces of Cisco Unified Communications Manager:

Cisco TAPI and Media Driver

Cisco Unified Communications Manager exposes sophisticated call control of IP telephony devices and soft-clients using the Computer Telephony TAPI interface. Cisco's Telephone Service Provider (TSP) and Media Driver interface enables custom applications to monitor telephony-enabled devices and call events, establish first-and third-party call control, and interact with the media layer to terminate media, play announcements, record calls.

For more information, see the TAPI and Media Driver Tech Center on the Cisco Developer Network at the following location:

http://developer.cisco.com/web/tapi/home

Cisco JTAPI

For more information, see the JTAPI Tech Center on the Cisco Developer Network at the following location:

http://developer.cisco.com/web/jtapi/home

Also, see Cisco Unified JTAPI Developers Guide for Cisco Unified Communications Manager for relevant release of Cisco Unified Communications Manager at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html

Cisco Web Dialer

The Web Dialer, which is installed on a Cisco Unified Communications Manager server, allows Cisco Unified IP Phone users to make calls from web and desktop applications. For example, the Web Dialer uses hyperlinked telephone numbers in a company directory to allow users to make calls from a web page by clicking the telephone number of the person that they are trying to call. The two main components of Web Dialer comprise the Web Dialer Servlet and the Redirector Servlet.

For more information, see the Web Dialer Tech Center on the Cisco Developer Network http://developer.cisco.com/web/wd/home.

For more information on Cisco Web Dialer, see Cisco Unified Communications Manager XML Developers Guide for relevant release of Cisco Unified Communications Manager at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Serviceability Interfaces

The following are the serviceability interfaces of Cisco Unified Communications Manager:

Serviceability XML

A collection of services and tools designed to monitor, diagnose, and address issues specific to Unified CM. serviceabiltiy XML interface:

- Provides platform, service and application performance counters to monitor the health of Unified CM hardware and software
- Provides real-time device and Computer Telephony Integration (CTI) connection status to monitor the health of phones, devices, and applications connected to Cisco Unified Communications Manager.
- Enables remote control (Start/Stop/Restart) of Cisco Unified Communications Manager services.
- Collects and packages Cisco Unified Communications Manager trace files and logs for troubleshooting and analysis.
- Provides applications with Call Detail Record files based on search criteria.
- Provides management consoles with SNMP data specific to Cisco Unified Communications Manager hardware and software.

For more information, see the Serviceability XML Tech Center on the Cisco Developer Network http://developer.cisco.com/web/sxml/home.

SNMP/MIBs

SNMP interface allows external applications to query and report various UCMgr entities. It provides information on the connectivity of the Unified Communication Manager to other devices in the network, including syslog information.

The MIBs supported by Cisco Unified Communications Manager includes:

- Cisco-CCM-MIB, CISCO-CDP-MIB, Cisco-syslog-MIB
- Standard MIBs like MIB II, SYSAPPL-MIB, HOST RESOURCES-MIB
- Vendor MIBs

For more information, see the SNMP/MIB Tech Center on the Cisco Developer Network http://developer.cisco.com/web/sxml/home.

Also, see Cisco Unified Communications Manager XML Developers Guide for relevant release of Cisco Unified Communications Manager at the following location:

http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html.

Routing Rules Interface

Cisco Unified Communication Manager 8.0(1) and later supports the external call control (ECC) feature, which enables an adjunct route server to make call-routing decisions for Cisco Unified Communications Manager by using the Cisco Unified Routing Rules Interface. When you configure external call control, Cisco Unified Communications Manager issues a route request that contains the calling party and called party information to the adjunct route server. The adjunct route server receives the request, applies appropriate business logic, and returns a route response that instructs Cisco Unified Communications Manager on how the call should get routed, along with any additional call treatment that should get applied.

For more information, see the Routing Rules Interface Tech Center on the Cisco Developer Network http://developer.cisco.com/web/curri/home.

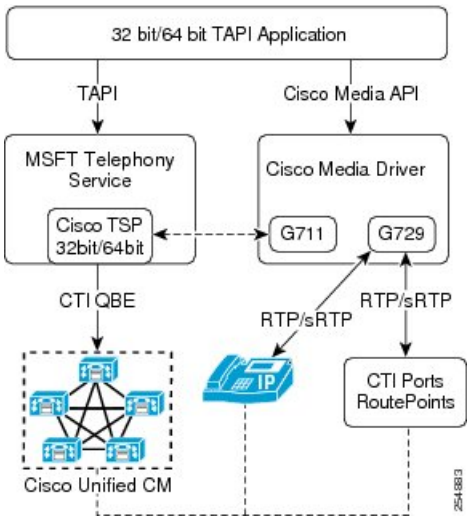
Cisco Unified TSP Overview

The standard TAPI provides an unchanging programming interface for different implementations. The goal of Cisco in implementing TAPI for the Cisco Unified Communications Manager platform remains to conform as closely as possible to the TAPI specification, while providing extensions that enhance TAPI and expose the advanced features of Cisco Unified Communications Manager to applications.

As versions of Cisco Unified Communications Manager and Cisco Unified TSP are released, variances in the API should be minor and should tend in the direction of compliance. Cisco stays committed to maintaining its API extensions with the same stability and reliability, though additional extensions may be provided as new Cisco Unified Communications Manager features become available.

The following figure shows the architecture of TAPI.

Figure 1: Architecture of TAPI Service Process



Note The Cisco TSP is a TAPI 2.1 service provider.

Cisco Unified TSP Concepts

The following are described in this section:

- Basic TAPI Applications, on page 5
- Cisco TSP Components, on page 6
- Cisco Media Drivers, on page 6
- TAPI Debugging, on page 6
- Cisco TSP Components, on page 6

See Basic TAPI Implementation, on page 139 and Cisco Device-Specific Extensions, on page 313 for lists and descriptions of interfaces and extensions.

Basic TAPI Applications

Microsoft has defined some basic APIs which can be invoked/supported from application code. All Microsoft defined APIs that can be used from the TAPI applications are declared in TAPI.H file. TAPI.H file is a standard library file that is with the VC++/VS2005 Installation. For example, C:\Program Files\Microsoft Visual Studio\VC98\Include\TAPI.H.

To use any specific API which is added or provided by Cisco TSP, the application needs to invoke that API by using the LineDevSpecific API.

Simple application

```
#include <tapi.h>#include <string>
#include "StdAfx.h"
class TapiLineApp {
LINEINITIALIZEEXPARAMS mLineInitializeExParams;//was declared in TAPI.h files
     HLINEAPP    mhLineApp;
     DWORD       mdwNumDevs;
     DWORD dwAPIVersion = 0x20005

public:
    // App Initialization
    // Note hInstance can be NULL
    // appstr - value can be given the app name "test program"
    bool TapiLineApp::LineInitializeEx(HINSTANCE hInstance, std::string appStr)
{
    unsigned long lReturn = 0;
    mLineInitializeExParams.dwTotalSize = sizeof(mLineInitializeExParams);
    mLineInitializeExParams.dwOptions = LINEINITIALIZEEXOPTION_USEEVENT;
    lReturn = lineInitializeEx (&mhLineApp, hInstance, NULL, appStr.c_str),
 &mdwNumDevs,&dwAPIVersion,&LineInitializeExParams);
    if ( lReturn = = 0 ) {
        return true;
    }
    else {
        return false;
    }
}
//App shutdown
bool TapiLineApp::LineShutdown()
{
    return! (lineShutdown (mhLineApp));
}
};
```

Cisco TSP Components

The following are Cisco TSP components:

- CiscoTSP dll– TAPI service implementation provided by Cisco TSP
- CTIQBE over TCP/IP – Cisco protocol used to monitor and control devices and lines
- CTI Manager Service – Manages CTI resources and connections to devices. Exposed to 3rd-party applications via Cisco TSP and/or JTAPI API

Cisco Media Drivers

Cisco Media Driver can be used to play announcements or record the call media. For information about the installation of the Media Drivers, see Cisco Media Driver Selection, on page 109.

TAPI Debugging

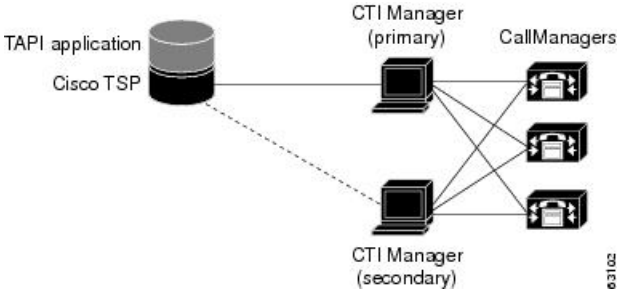
The TAPI browser is a TAPI debugging application. It can be downloaded from the Microsoft MSDN Web site at ftp://ftp.microsoft.com/developr/TAPI/tb20.zip. The TAPI browser can be used to initialize TAPI, for use by TAPI developers to test a TAPI implementation and to verify that the TSP is operational.

CTI Manager (Cluster Support)

The CTI Manager, along with the Cisco Unified TSP, provide an abstraction of the Cisco Unified Communications Manager cluster that allows TAPI applications to access Cisco Unified Communications Manager resources and functionality without being aware of any specific Cisco Unified Communications Manager. The Cisco Unified Communications Manager cluster abstraction also enhances the failover capability of CTI Manager resources. A failover condition occurs when a node fails, a CTI Manager fails, or a TAPI application fails, as illustrated in the following figure.

Note Cisco does not support CTI device monitoring or call control with 3rd-party devices.

Figure 2: Cluster Support Architecture



Cisco Unified Communications Manager Failure

When a Cisco Unified Communications Manager node in a cluster fails, the CTI Manager recovers the affected CTI ports and route points by reopening these devices on another Cisco Unified Communications Manager node. When the failure is first detected, Cisco Unified TSP sends a PHONE_STATE (PHONESTATE_SUSPEND) message to the TAPI application.

When the CTI port/route point is successfully reopened on another Cisco Unified Communications Manager, Cisco Unified TSP sends a phone PHONE_STATE (PHONESTATE_RESUME) message to the TAPI application. If no Cisco Unified Communications Manager is available, the CTI Manager waits until an appropriate Cisco Unified Communications Manager comes back in service and tries to open the device again. The lines on the affected device also go out of service and in service with the corresponding LINE_LINEDEVSTATE (LINEDEVSTATE_OUTOFSERVICE) and LINE_LINEDEVSTATE (LINEDEVSTATE_INSERVICE) events Cisco Unified TSP sends to the TAPI application. If for some reason the device or lines cannot be opened, even when all Cisco Unified Communications Managers come back in service, the system closes the devices or lines, and Cisco Unified TSP will send PHONE_CLOSE or LINE_CLOSE messages to the TAPI application.

When a failed Cisco Unified Communications Manager node comes back in service, CTI Manager "re-homes" the affected CTI ports or route points to their original Cisco Unified Communications Manager. The graceful re-homing process ensures that the re-homing only starts when calls are no longer being processed or are active on the affected device. For this reason, the re-homing process may not finish for a long time, especially for route points, which can handle many simultaneous calls.

When a Cisco Unified Communications Manager node fails, phones currently re-home to another node in the same cluster. If a TAPI application has a phone device opened and the phone goes through the re-homing process, CTI Manager automatically recovers that device, and Cisco Unified TSP sends a PHONE_STATE (PHONESTATE_SUSPEND) message to the TAPI application. When the phone successfully re-homes to

another Cisco Unified Communications Manager node, Cisco Unified TSP sends a PHONE_STATE (PHONESTATE_RESUME) message to the TAPI application.

The lines on the affected device also go out of service and in service, and Cisco Unified TSP sends LINE_LINEDEVSTATE (LINEDEVSTATE_OUTOFSERVICE) and LINE_LINEDEVSTATE (LINEDEVSTATE_INSERVICE) messages to the TAPI application.

Call Survivability

When a device or Cisco Unified Communications Manager failure occurs, no call survivability exists; however, media streams that are already connected between devices will survive. Calls in the process of being set up or modified (transfer, conference, redirect) simply get dropped.

CTI Manager Failure

When a primary CTI Manager fails, Cisco Unified TSP sends a PHONE_STATE (PHONESTATE_SUSPEND) message and a LINE_LINEDEVSTATE (LINEDEVSTATE_OUTOFSERVICE) message for every phone and line device that the application opened. Cisco Unified TSP then connects to a backup CTIManager. When a connection to a backup CTI Manager is established and the device or line successfully reopens, the Cisco Unified TSP sends a PHONE_STATE (PHONESTATE_RESUME) or LINE_LINEDEVSTATE (LINEDEVSTATE_INSERVICE) message to the TAPI application. If the Cisco Unified TSP is unsuccessful in opening the device or line for a CTI port or route point, the Cisco Unified TSP closes the device or line by sending the appropriate PHONE_CLOSE or LINE_CLOSE message to the TAPI application.

After Cisco Unified TSP is connected to the backup CTIManager, Cisco Unified TSP will not reconnect to the primary CTIManager until the connection is lost between Cisco Unified TSP and the backup CTIManager.

If devices are added to or removed from the user while the CTI Manager is down, Cisco Unified TSP generates PHONE_CREATE/LINE_CREATE or PHONE_REMOVE/LINE_REMOVE events, respectively, when connection to a backup CTI Manager is established.

Cisco Unified TAPI Application Failure

When a Cisco TAPI application fails (the CTI Manager closes the provider), calls at CTI ports and route points that have not yet been terminated get redirected to the Call Forward On Failure (CFF) number that has been configured for them. The system routes new calls into CTI Ports and Route Points that are not opened by an application to their CFNA number.

LINE_CALLDEVSPECIFIC Event Support for RTP Events

RTP events are generated as LINE_CALLDEVSPECIFIC events that contain Call Handle details of the call. However, to activate the feature, the application must negotiate the extension version greater than or equal to 0x00040001 when opening the line.

Due to dependency on the extension version of the line, the Media Events, RTP_START / STOP, are reported differently to the application:

- If extension version is less than EXTVERSION_FOUR_DOT_ZERO - 0x00040000 — TSP reports LINE_DEVSPECIFIC event to application on the line irrespective whether call object is present. In this case, even if a call is DeAllocated after IDLE state, RTP_STOP events are delivered to the application.
- If extension version is greater than or same as EXTVERSION_FOUR_DOT_ZERO - 0x00040000—TSP does report the Media Events if the Call Object is DeAllocated from Application.

So a check must be added for the Extension Version to maintain backward compatibility. So it must not be assumed that RTP events will always come before IDLE event.

QoS

Cisco Unified TSP supports the Cisco baseline for baselining of Quality of Service (QoS). Cisco Unified TSP marks the IP Differentiated Services Code Point (DSCP) for QBE control signals that flow from TSP to CTI with the value of the Service parameter "DSCP IP for CTI Applications" that CTI sends in the ProviderOpenCompletedEvent. The Cisco TAPI Wave driver marks the RTP packets with the value that CTI sends in the StartTransmissionEvent. The system stores the DSCP value received in the StartTransmissionEvent in the DevSpecific portion of the LINECALLINFO structure, and fires the LINECALLINFOSTATE_DEVSPECIFIC event with the QoS indicator.

Note QoS information is not available if you begin monitoring in the middle of a call because existing calls do not have an RTP event.

Presentation Indication (PI)

There is a need to separate the presentability aspects of a number (calling, called, and so on) from the actual number itself. For example, when the number is not to be displayed on the IP phone, the information might still be needed by another system, such as Unity VM. Hence, each number/name of the display name needs to be associated with a Presentation Indication (PI) flag, which will indicate whether the information should be displayed to the user or not.

You can set up this feature as follows:

On a Per-Call Basis

You can use Route Patterns and Translation Patterns to set or reset PI flags for various partyDNs/Names on a per-call basis. If the pattern matches the digits, the PI settings that are associated with the pattern will be applied to the call information.

On a Permanent Basis

You can configure a trunk device with "Allow" or "Restrict" options for parties. This will set the PI flags for the corresponding party information for all calls from this trunk.

Cisco Unified TSP supports this feature. If calls are made via Translation patterns with all of the flags set to Restricted, the system sends the CallerID/Name, ConnectedID/Name, and RedirectionID/Name to applications as Blank. The system also sets the LINECALLPARTYID flags to Blocked if both the Name and Party number are set to Restricted.

When developing an application, be sure only to use functions that the Cisco TAPI Service Provider supports. For example, the Cisco TAPI Service Provider supports transfer, but not fax detection. If an application requires an unsupported media or bearer mode, the application will not work as expected.

Cisco Unified TSP does not support TAPI 3.0 applications.

Call Control

You can configure Cisco Unified TSP to provide first-or third-party call control.

First-Party Call Control

In first-party call control, the application terminates the audio stream. Ordinarily, this occurs by using the Cisco wave driver. However, if you want the application to control the audio stream instead of the wave driver, use the Cisco device-specific extensions.

Third-Party Call Control

In third-party call control, the control of an audio stream terminating device is not "local" to the Cisco Unified Communications Manager. In such cases, the controller might be the physical IP phone on your desk or a group of IP phones for which your application is responsible.

Note Cisco does not support CTI device monitoring or call control with 3rd-party devices.

CTI Port

For first-party call control, a CTI port device must exist in the Cisco Unified Communications Manager. Because each port can only have one active audio stream at a time, most configurations only need one line per port.

A CTI port device does not actually exist in the system until you run a TAPI application and a line on the port device is opened requesting LINEMEDIAMODE_AUTOMATEDVOICE and LINEMEDIAMODE_INTERACTIVEVOICE. Until the port is opened, anyone who calls the directory number that is associated with that CTI port device receives a busy or reorder tone.

The IP address and UDP port number is either specified statically (the same IP address and UDP port number is used for every call) or dynamically. By default, CTI ports use static registration.

Dynamic Port Registration

Dynamic Port Registration enables applications to specify the IP address and UDP port number on a call-by-call basis. Currently, the IP address and UDP port number are specified when a CTI port registers and is static through the life of the registration of the CTI port. When media is requested to be established to the CTI port, the system uses the same static IP address and UDP port number for every call.

An application that wants to use Dynamic Port Registration must specify the IP address and UDP port number on a call before invoking any features on the call. If the feature is invoked before the IP address and UDP port number are set, the feature will fail, and the call state will be set depending on when the media time-out occurs.

CTI Route Point

You can use Cisco Unified TAPI to control CTI route points. CTI route points allow Cisco Unified TAPI applications to redirect incoming calls with an infinite queue depth. This allows incoming calls to avoid busy signals.

CTI route point devices have an address capability flag of LINEADDRCAPFLAGS_ROUTEPOINT. When your application opens a line of this type, it can handle any incoming call by disconnecting, accepting, or redirecting the call to some other directory number. The basis for redirection decisions can be caller ID information, time of day, or other information that is available to the program.

Media Termination at Route Point

The Media Termination at Route Point feature lets applications terminate media at route points. This feature enables applications to pass the IP address and port number where they want the call at the route point to have media established.

The system supports the following features at route points:

- Answer
- Multiple Active Calls
- Redirect
- Hold
- UnHold
- Blind Transfer
- DTMF Digits
- Tones

Monitoring Call Park Directory Numbers

The Cisco Unified TSP supports monitoring calls on lines that represent Call Park Directory Numbers (Call Park DNs). The Cisco Unified TSP uses a device-specific extension in the LINEDEVCAPS structure that allows TAPI applications to differentiate Call Park DN lines from other lines. If an application opens a Call Park DN line, all calls that are parked to the Call Park DN get reported to the application. The application cannot perform any call control functions on any calls at a Call Park DN.

To open Call Park DN lines, you must check the **Monitor Call Park DNs** check box in Cisco Unified Communications Manager User Administration for the Cisco Unified TSP user. Otherwise, the application will not perceive any of the Call Park DN lines upon initialization.

Multiple Cisco Unified TSPs

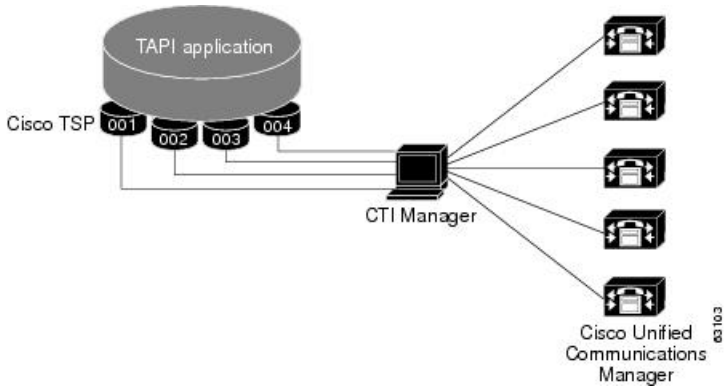
In the Cisco Unified TAPI solution, the TAPI application and Cisco Unified TSP get installed on the same machine. The Cisco Unified TAPI application and Cisco Unified TSP do not directly interface with each other. A layer written by Microsoft sits between the TAPI application and Cisco Unified TSP. This layer, known as TAPISRV, allows the installation of multiple TSPs on the same machine, and it hides that fact from the Cisco Unified TAPI application. The only difference to the TAPI application is that it is now informed that there are more lines that it can control.

Consider an example—assume that Cisco Unified TSP1 exposes 100 lines, and Cisco Unified TSP2 exposes 100 lines. In the single Cisco Unified TSP architecture where Cisco Unified TSP1 is the only Cisco Unified TSP that is installed, Cisco Unified TSP1 would tell TAPISRV that it supports 100 lines, and TAPISRV would tell the application that it can control 100 lines. In the multiple Cisco Unified TSP architecture, where both

Cisco Unified TSPs are installed, this means that Cisco Unified TSP1 would tell TAPISRV that it supports 100 lines, and Cisco Unified TSP2 would tell TAPISRV that it supports 100 lines. TAPISRV would add the lines and inform the application that it now supports 200 lines. The application communicates with TAPISRV, and TAPISRV takes care of communicating with the correct Cisco Unified TSP.

Ensure that each Cisco Unified TSP is configured with a different username and password that you administer in the Cisco Unified Communications Manager Directory. Configure each user in the Directory, so devices that are associated with each user do not overlap. Each Cisco Unified TSP in the multiple Cisco Unified TSP system does not communicate with the others. Each Cisco Unified TSP in the multiple Cisco Unified TSP system creates a separate CTI connection to the CTI Manager as shown in the following figure. Multiple Cisco Unified TSPs help in scalability and higher performance.

Figure 3: Multiple Cisco Unified TSPs Connect to CTI Manager



CTI Device/Line Restriction

With CTI Device/Line restriction implementation, a CTIRestricted flag is be placed on device or line basis. When a device is restricted, it assumes that all its configured lines are restricted.

Cisco Unified TSP does not report any restricted devices and lines back to application. When a CTIRestricted flag is changed from Cisco Unified Communications Manager Administration, Cisco Unified TSP treats it as normal device/line add or removal.

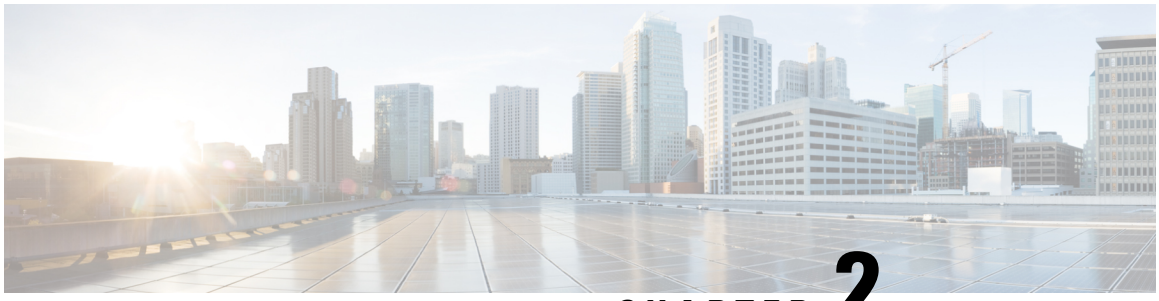
Development Guidelines

Cisco maintains a policy of interface backward compatibility for at least one previous major release of Cisco Unified Communications Manager. Cisco still requires Cisco Technology Developer Program member applications to be retested and updated as necessary to maintain compatibility with each new major release of Cisco Unified Communications Manager.

The following practices are recommended to all developers, including those in the Cisco Technology Developer Program, to reduce the number and extent of any updates that may be necessary:

- The order of events and/or messages may change. Developers should not depend on the order of events or messages. For example, where a feature invocation involves two or more independent transactions, the events or messages may be interleaved. Events related to the second transaction may precede messages related to the first. Additionally, events or messages can be delayed due to situations beyond control of the interface (for example, network or transport failures). Applications should be able to recover from out of order events or messages, even when the order is required for protocol operation.

- The order of elements within the interface event or message may change, within the constraints of the protocol specification. Developers must avoid unnecessary dependence on the order of elements to interpret information.
- New interface events, methods, responses, headers, parameters, attributes, other elements, or new values of existing elements, may be introduced. Developers must disregard or provide generic treatments where necessary for any unknown elements or unknown values of known elements encountered.
- Previous interface events, methods, responses, headers, parameters, attributes, and other elements, will remain, and will maintain their previous meaning and behavior to the extent possible and consistent with the need to correct defects.
- Applications must not be dependent on interface behavior resulting from defects (behavior not consistent with published interface specifications) since the behavior can change when defect is fixed.
- Use of deprecated methods, handlers, events, responses, headers, parameters, attributes, or other elements must be removed from applications as soon as possible to avoid issues when those deprecated items are removed from Cisco Unified Communications Manager.
- Application Developers must be aware that not all new features and new supported devices (for example, phones) will be forward compatible. New features and devices may require application modifications to be compatible and/or to make use of the new features/devices.



CHAPTER 2

New and Changed Information

This chapter describes new and changed Cisco Unified TAPI Service Provider (TSP) information for Cisco Unified Communications Manager release 10.0(1) and features supported in the previous releases.

Refer to the programming guides Web site for prior Cisco TAPI Developer Guides at http://www.cisco.com/en/US/products/sw/voicesw/ps556/products_programming_reference_guides_list.html

This chapter contains the following sections:

Cisco Unified Communications Manager Release 10.0(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 10.0(1) and contains the following topics:

Features Supported in Previous Releases

This section describes the features supported in previous releases and contains the following sections:

Cisco Unified Communications Manager Release 9.1(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 9.1(1) and contains the following topics:

CTI Remote Device:

Cisco Unified Communications Manager Release 9.0(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager release 9.0(1) and contains the following topics:

Cisco Unified Communications Manager Release 8.6(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager release 8.6(1) and contains the following topics:

- EnergyWise DeepSleep Mode Support, on page 60
- FIPS Compliance, on page 65
- Password Expiry Notification, on page 80

Cisco Unified Communications Manager Release 8.5(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager release 8.5(1) and contains the following topics:

- Agent Greeting, on page 26
- Agent Zip Tone, on page 27
- Early Offer, on page 55
- Extension Mobility Memory Optimization Option, on page 63
- Other-Device State Notification, on page 77
- Unrestricted Unified CM, on page 98
- Whisper Coaching, on page 100

Note Cisco announces the end-of-Availability for Cisco TAPI Wave Driver. The last release includes the affected product(s) is Unified Communication Manager Release 8.X. Unified Communication Manager 9.0(1) and later does not include Cisco TAPI Wave Driver.

Note Develop Partners are encouraged to adopt support for Cisco TAPI Media Driver introduced in Unified Communication Manager 8.0(1) (released March 2010), which provides like or better functionality. Information for Cisco TAPI Media Driver can be found in the TAPI Developer Guides at the following location:

http://www.cisco.com/en/US/partner/products/sw/voicesw/ps556/products_programming_reference_guides_list.html

Cisco Unified Communications Manager Release 8.0(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 8.0(1) and contains the following topics:

- Call Control Discovery, on page 28
- Call PickUp, on page 29
- CallFwdAll Notification, on page 34

- End-to-End Call Trace, on page 60
- Extension Mobility Cross Cluster, on page 62
- External Call Control, on page 64
- Hunt List, on page 68
- New Cisco Media Driver, on page 76
- Secured Monitoring and Recording, on page 88
- Support for 100 + Directory Numbers, on page 96

Cisco Unified Communications Manager Release 7.1(3)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 7.1(3) and contains the following topics:

- Support for Cisco Unified IP Phone 6900 and 9900 Series, on page 93

Cisco Unified Communications Manager Release 7.1(2)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 7.1(2) and contains the following topics:

- IPv6, on page 70
- Direct Transfer Across Lines, on page 52
- Message Waiting Indicator Enhancement, on page 75
- Swap and Cancel Softkeys, on page 96
- Drop-Any-Party, on page 54
- Park Monitoring, on page 78
- Logical Partitioning, on page 74
- Support for Cisco Unified IP Phone 6900 and 9900 Series, on page 93
- Device State Server, on page 50

Cisco Unified Communications Manager Release 7.0(1)

This section describes new and changed features supported in Cisco Unified Communications Manager Release 7.0(1) and contains the following:

- Join Across Lines (SIP), on page 72
- Localization Infrastructure Changes, on page 73
- Secure Conference, on page 83
- Calling Party Normalization, on page 29

- Click to Conference, on page 36
- Microsoft Windows Vista, on page 75

Note For the features, Join Across Lines, Do Not Disturb-Reject, and Calling Party Normalization, each TAPI application must be upgraded to a version that is compatible with these features. Additionally, if you are upgrading from Release 5.1 and you use Join Across Lines, the Conference Chaining feature must not be enabled or used until all applications are either upgraded to a version compatible with the new CUCM version. Also, you should verify that the applications are not impacted by the Conference Chaining feature.

Cisco Unified Communications Manager Release 6.1(x)

This section describes new and changed features that Cisco Unified Communications Manager Release 6.1(x) supports and contains the following topic:

- Join Across Lines (SCCP), on page 72

Cisco Unified Communications Manager Release 6.0(1)

This section describes new and changed features that are supported in Cisco Unified Communications Manager Release 6.0(1), and contains the following topics:

- Intercom, on page 69
- Secure Conference, on page 83
- Do Not Disturb, on page 53
- Conference Enhancements, on page 37
- Arabic and Hebrew Language, on page 28
- Additional Features Supported on SIP Phones, on page 25
- Silent Monitoring, on page 90
- Call Recording and Call Recording Enhancement, on page 32
- Calling Party IP Address, on page 29

Cisco Unified Communications Manager Release 5.1

This section describes new and changed features supported in Cisco Unified Communications Manager, Release 5.1 and contains the following topics:

- Partition, on page 79
- Alternate Script, on page 28
- Secure RTP, on page 84
- Presentation Indication, on page 98

- Refer and Replaces for Phones That Are Running SIP, on page 83
- SIP URL Address, on page 91
- Presentation Indication, on page 98
- Presentation Indication, on page 98
- Unicode, on page 98

Cisco Unified Communications Manager Release 5.0

This section describes new and changed features that are supported in Cisco Unified Communications Manager, Release 5.0, and contains the following topics:

- Unicode, on page 98
- Line-Side Phones That Run SIP, on page 73

Cisco Unified Communications Manager Release 4.x

This section describes new and changed features that are supported in Cisco Unified Communications Manager, Release 4.x, and contains the following topics:

Release 4.0

- Redirect and Blind Transfer, on page 81
- Direct Transfer, on page 52
- Join, on page 71
- Set the Original Called Party Upon Redirect, on page 89
- Cisco Unified TSP Auto Update, on page 34
- Shared Line Appearance, on page 89
- XSI Object Pass Through, on page 103

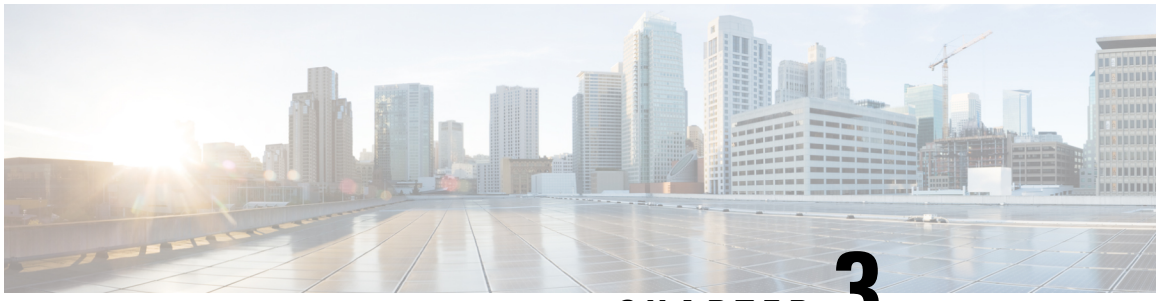
Release 4.1

- Forced Authorization Code and Client Matter Code, on page 65
- CTI Port Third-Party Monitoring Port, on page 38
- Translation Pattern, on page 98
- Hold Reversion, on page 67

Cisco Unified Communications Manager Releases Prior to 4.x

The chapter includes the following list of all features that are available in the Cisco Unified TSP implementation of Cisco Unified Communications Manager, prior to Release 4.x:

- Forwarding, on page 66
- Extension Mobility, on page 62
- Privacy Release, on page 81
- Join, on page 71
- Privacy Release, on page 81
- Barge and cBarge, on page 28
- XSI Object Pass Through, on page 103
- Silent Install, on page 90



CHAPTER 3

Features Supported by TSP

This chapter describes the features that Cisco Unified TAPI Service Provider (TSP) supports. See New and Changed Information, on page 15 for described features, which are listed by Cisco Unified Communications Manager release.

3XX

Cisco TSP maps the CTI reason code for 3XX to REDIRECT. When a call arrives on a monitored line due to 3XX feature, the call reason for the incoming call will get REDIRECT in this case. No interface change for TSP 3XX support.

Backward Compatibility

This feature is backward compatible.

Additional Features Supported on SIP Phones

Cisco Unified Communications Manager extends support for phones that are running SIP with these new features:

- PhoneSetLamp (but only for setting the MWI lamp)
- PhoneSetDisplay
- PhoneDevSpecific (CPDST_SET_DEVICE_UNICODE_DISPLAY)
- LineGenerateTone
- Park and UnPark
- The LINECALLREASON_REMINDER reason for CallPark reminder calls
- PhoneGetDisplay (but only after a PhoneSetDisplay)

TSP does not pass unicode name for phones that are running SIP.

Agent Greeting

Agent Greeting allows a CTI application (for example, the Contact Center) to instruct Cisco Unified Communications Manager to automatically play a pre-recorded announcement to the customer immediately after a successful media connection to the agent device. Applications are responsible for answering this call and playing greeting (for example, “Thank you for calling Citibank Visa. My name is Joe. May I have your account number please.”). The agent and customer can hear the agent greeting and the agent can remain on mute until the greeting ends or talk to the customer before the greeting ends. To support the Agent Greeting feature, Cisco Unified TSP introduces a new CCiscoLineDevSpecific extension to initiate and stop Agent Greeting.

To handle agent greetings during the customer calls:

- CCiscoLineDevSpecificStartSendMediaToBIBRequest allows the application to initiate an agent greeting to the customer call. The request contains IVRDN and CGPNToIVR.
- CCiscoLineDevSpecificStopSendMediaToBIBRequest allows the application to stop an agent greeting. This request is placed on the line where the agent greeting is currently playing and no other parameter is required.
- When an agent greeting is initiated or stopped, Cisco Unified TSP sends LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) or LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED) to application.
- In the case of an agent greeting ended event, param2 indicates if the agent greeting was played successfully.
- If the application opens another line while an agent greeting is currently playing to the agent-to-customer call, Cisco Unified TSP exposes SendMediaToBIB bitmap in CallAttributeBits bitmap of LineCallInfo::DevSpecific indicating that the agent greeting is in progress. When the agent greeting ends, Cisco Unified TSP sends LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED) to the application and clears SendMediaToBIB bitmap in the CallAttributeType field.
- When the agent greeting call arrives on the IVR port, two new bitmaps (ServerCall and BIBCall) are exposed as CallAttributeType. Cisco Unified TSP exposes LINECALLREASON_UNKNOWN as TAPI call reason when the agent greeting call arrives at IVR port, and exposes CtiReasonSendMediaToBIB in the ExtendedCallReason field.
- CTI Port and Route Point do not support Agent Greeting. Application gets LINEERR_OPERATIONUNAVAIL if CCiscoLineDevSpecificStartSendMediaToBIBRequest is issued on a CTI port or Route Point.

Note To support this feature, the application must negotiate line extension 0x000B0000 or above.

Interface Changes

See Start Send Media to BIB, on page 376, Stop Send Media to BIB, on page 377, Media to BIB Started Event, on page 407, Media to BIB Ended Event, on page 408, and Details, on page 326.

Message Sequences

See Agent Greeting, on page 445.

Backward Compatibility

This feature is backward compatible.

Agent Zip Tone

The Agent Zip Tone Support feature allows the TAPI applications to play tone on active calls, that is, play a tone on an agent phone after the call is answered (active state) by an agent.

TAPI defines a new line devspecific CCiscoLineDevSpecificPlaytone (lineHandle, callHandle, Tone, and PlayToneDirection) message type. Applications use this message type to play the tone on a phone placed locally or remotely.

Applications can play the tone on a local phone when the call is in the Accepted/Ringback state and on a remote phone when the call is in the Connected state.

When the application issues CCiscoLineDevSpecificPlaytone request with tone as CTONE_ZIP and direction as local, then the Zip tone is played at the local phone and the

LINE_DEVSPECIFIC event with the following parameters is reported on the local phone indicating that the tone is played:

- dwParam1 = SLDSMT_CALL_TONE_CHANGED
- dwParam2 = CTONE_ZIP
- dwParam3 = 0(local)

Similarly, when the application issues CCiscoLineDevSpecificPlaytone request with tone as CTONE_ZIP and direction as remote, then the Zip tone is played at the remote phone and the

LINE_DEVSPECIFIC event the following parameters is reported on the remote phone indicating that the tone is played:

- dwParam1 = SLDSMT_CALL_TONE_CHANGED
- dwParam2 = CTONE_ZIP
- dwParam2 = 0(local)

and also LINE_DEVSPECIFIC event the following parameters is reported on the local phone:

- dwParam1 = SLDSMT_CALL_TONE_CHANGED
- dwParam2 = CTONE_ZIP
- dwParam3 = 1(remote)

Extension 0x000B0000 is introduced for release 8.5(1).

Interface Changes

See Agent Zip Tone, on page 462.

Message Sequences

See Agent Zip Tone, on page 462.

Backward Compatibility

This feature is backward compatible.

Alternate Script

Certain IP phone types support an alternate language script other than the default script that corresponding to the phone configurable locale. For example, the Japanese phone locale associates two written scripts. Some phone types support only the default Katakana script, while other phones types support both the default Katakana script and the alternate Kanji script. Because applications can send text information to the phone for display purposes, they need to know what alternate script a phone supports – if any.

Arabic and Hebrew Language

Users can select Arabic and Hebrew languages during installation and also in the Cisco TSP settings user interface.

Barge and cBarge

Cisco Unified Communications Manager supports the Barge and cBarge features. The Barge feature uses the built-in conference bridge. The cBarge feature uses the shared conference resource.

Cisco Unified TSP supports the events that are caused by the invocation of the Barge and cBarge features. It does not support invoking either Barge or cBarge through an API of Cisco Unified TSP.

Call Control Discovery

The Call Control Discovery feature facilitates provisioning for inter-call agent communications. It uses the Service Advertisement Framework (SAF) network service to advertise itself as a call control entity and to discover other call control entities (CUCMs or CMEs) on the network so that it can dynamically adapt their routing behavior.

When call is made between two devices on different clusters, and the ICT bandwidth doesn't allow the call to go through, the CCD feature will fail over the call through a PSTN trunk to reach the same destination. PSTN failover will also be triggered by the CCDRequestingService when the call to a learned Hosted DNPattern gets rejected with a cause code other than unallocated, unassigned number and user busy

TAPI shall pass the CtiReasonSAF_CCD_PSTNFailover in the existing ExtendedCallReason field of the devSpecific portion of lineCallInfo. Since TAPI requires the TAPI call reason field to be set only once, the TAPI call reason shall remain as LINECALLREASON_DIRECT.

Interface Changes

There are no interface changes.

Message Sequences

See Call Control Discovery, on page 475

Backward Compatibility

This feature is backward compatible.

Calling Party IP Address

The Calling Party IP Address feature provides the IP address of the calling party. The calling party device, which must be supported, must be an IP phone. The IP address is provided to applications in the devspecific data of LINECALLINFO. A value of zero (0) indicates that the information in not available.

The enhancement provides the IP address to the destination side of basic calls, consultation calls for transfer and conference, and basic redirect and forwarding. If the calling party changes, no support is provided.

Message Sequence

See Calling Party IP Address, on page 497.

Calling Party Normalization

Prior to the Cisco Unified Communication Manager Release 7.0(1), the "+" symbol was not supported. Also, no support existed for displaying the localized or global number of the caller to the called party on its alerting display and the entry into its call directories for supporting a callback without the need of an EditDial.

Cisco Unified Communications Manager Release 7.0(1) adds support for "+" symbol and also the calling number is globalized and passed to the application. This enables the end user to dial back without using EditDial. Along with the globalized calling party, the user would also get the number type of the calling party. This would help the user to know where the call originated, that is, whether it is a SUBSCRIBER, NATIONAL or INTERNATIONAL number.

Interface Changes

See LINECALLINFO, on page 320.

Message Sequences

See Calling Party Normalization, on page 498.

Backward Compatibility

This feature is backward compatible.

Call PickUp

Call Pickup enables TAPI applications to invoke pickup, group-pickup, other-pickup, and directed pickup features from the application. Apart from providing the API to invoke Call PickUp feature, application registers

Call pickup groups for alert notification, whenever a call is available for pickup. There will not be any notification if the call is picked up and the alerting stops.

Whenever there is a new call on the Pickup Group, TAPI fires a LINE_APPNEWCALL event followed by a LINE_CALLSTATE with a LINECALLSTATE_UNKNOWN | CLDSMT_CALL_PICKUP_STATE.

TAPI provides the pickup group Direct Number or Partition for the line in the devSpecific data of LINEDEVCAPS when LineGetDevCaps API is invoked with Extension version 0x000A0000 or higher.

New LineType is added for this feature, which is exposed to Application in Devspecific part of LINEDEVCAPS for the Pickup Line.

#define LINEDEVCAPSDEVSPECIFIC_PICKUPDN 0x00000004

New extension 0x000A0000 must be negotiated to use the new APIs.

Range of Permanent Line ID for the Pickup Line is between MAX_PICKUP_PERMID and MIN_PICKUP_PERMID.

const DWORD MAX_PICKUP_PERMID = 0xFFFFFFFF;

const DWORD MIN_PICKUP_PERMID = 0xFF000000;

New Call State Callpickup State is added for this feature

#define CLDSMT_CALL_PICKUP_STATE 0x10000000

Interface Changes

See RegisterCallPickUpGroupForNotification, on page 374, UnRegisterCallPickUpGroupForNotification, on page 375, and CallPickUpRequest, on page 375.

Message Sequences

See Call PickUp, on page 501

Backward Compatibility

This feature is backward compatible.

Call Queuing Feature Support

Cisco Unified Communications Manager queuing feature provides the ability to hold callers in a queue if there are more calls distributed through the call distribution feature than it can handle at any given time until the hunt members are available to answer them. While a call is in queue, the user is given the initial greeting announcement, music on hold, repeated announcements and so on.

TAPI exposes the below call reasons for the respective conditions in extendedCallInfo in the DevSpecific part of the LINECALLINFO structure.

- CallQueue {45 (0x2D)}

  CallQueue call reason is exposed on the calling party call when the call is queued when all HuntMembers of the HuntGroup are busy

- CallDeQueue {46 (0x2E)}

CallDeQueue call reason is exposed on the connected party call when the dequeued-call is offered on the available Huntmember or any other DN which has been configured in the Queuing feature if no Huntmember is available

- CallDeQueueTimerExpired {47 (0x2F)}

  CallDeQueueTimerExpired call reason is exposed on the connected party call when the queued-call timer expires and the call is offered on the DN which has been configured in the Queuing feature when the "Maximum wait time in Queue" expires

- CallDeQueueAgentsBusy {48 (0x30)}

  CallDeQueueAgentsBusy call reason is exposed on the connected party call when all the hunt members are busy and hence the call is never queued and directly offered on the DN which has been configured in the Queuing feature when "Maximum Number of callers allowed in Queue" is reached

- CallDeQueueAgentsUnavailable {49 (0x31)}

  CallDeQueueAgentsUnavailable call reason is exposed on the connected party call when no hunt member is either logged-in or registered and the call is offered on the DN which has been configured in the Queuing feature when "No hunt members are Logged-in or registered"

TAPI shall not expose the ConnectedHuntPilotDN in the devspecific part of LINECALLINFO, on the calling party when the call is queued. The connectedHuntPilotDN is updated when the de-queued call offered on one of the hunt members is answered and goes to connected state.

The following Call queue setting configurations are available in the Hunt Pilot Configuration page.

Maximum Number of Callers Allowed in Queue (1-100): This is the queue depth configuration and reflects the maximum number calls that can be in the queue at any point of time.

Destination When Queue is full: It is the user configurable destination number to which the calls are forwarded when the maximum number of calls allowed in queue limit is reached.

Maximum Wait Time in Queue (10 -3600 seconds): User configurable maximum wait time a call on be in the queue.

Destination When Maximum Wait Time is Met: User configurable destination DN to which the call is forwarded when the maximum wait time in queue is reached.

Destination When There Are No Agents Logged In or Registered: User configurable destination DN to which the queue feature forwards the calls when none of the hunt members in the HuntPilot are registered or logged in.

Interface Changes

Not applicable.

Message Sequences

See Call Queuing, on page 508.

Backward Compatibility

This feature is not backward compatible.

Call Recording and Call Recording Enhancement

The Call Recording feature provides two ways of recording the conversations between the agent and the customer: automatic call recording and selective call recording. A line appearance configuration determines which mode is enabled. Administrators can configure no recording, automatic recording of all calls, or selective recording for a line appearance. In selective call recording, recording can be initiated using a softkey or programmable line key assigned to the device, a CTI-enabled application, or both interchangeably.

Selective recording supports two modes: silent recording and user recording.

In the silent recording mode, the call recording status is not reflected on the Cisco IP device display. Silent recording is typically used in a call center environment to enable a supervisor to record an agent call. A CTI-enabled application running on the supervisor desktop is generally used to start and stop the recording for the agent-customer call.

In the user recording mode, the call recording status is reflected on the Cisco IP device display. A recording may be started or stopped using a softkey, programmable line key, or CTI-enabled application running on the user desktop.

The recording configuration on a line appearance cannot be overridden by an application. TSP will report 'Recording type' information to app in devSpecificData of LineDevCaps structure. Whenever there is a change in 'Recording Type', TSP will send LINE_DEVSPECIFIC (SLDSMT_LINE_PROPERTY_CHANGED with indication of LPCT_RECORDING_TYPE) event to application.

If the automatic call recording is enabled, a recording session will be triggered whenever a call is received or initiated from the line appearance. When the application invoked call recording is enabled, application can start a recording session by using CCiscoLineDevSpecificStartCallRecording (SLDST_START_CALL_RECORDING) on the call after it becomes active. The selective recording can occur in the middle of the call, whereas the automatic recording always starts at the beginning of the call. The recorder is configured in CallManager as a SIP trunk device. Recorder DN can not be overridden by an application.

TSP will provide start recording request in lineDevSpecific to app for establishing a recording session. Application need to provide toneDirection as input to TSP in the start recording request. The result of the recording session is that the two media streams of the recorded call (agent-customer call) is being relayed from agent's phone to the recorder. TSP will provide agent's CCM Call Handle in the devSpecificData of LINECALLINFO.

TSP will inform the application when recording starts on its call by sending LINE_CALLDEVSPECIFIC (SLDSMT_RECORDING_STARTED) event. TSP will provide recording call attribute information (deviceName, DN, Partition) in devspecific data of LINECALLINFO after recording starts.

The recording session will be terminated when the call is ended or if app sends stop recording request to TSP through lineDevSpecific – CciscoLineDevSpecificStopCallRecording (SLDST_STOP_CALL_RECORDING).TSP will inform agent by sending LINE_CALLDEVSPECIFIC (SLDSMT_RECORDING_ENDED) when recording is stopped by stop recording request.

Both recording and monitoring get supported only for IP phones/CTI supported phones that are running SIP and within one cluster. It can be invoked only on phones that support built in bridges. Also built in bridge should be turned on to monitor or record calls on a device. Monitoring party does not need to have a BIB configured. Recording and monitoring will not be supported for secure calls in this phase.

Call Attributes

Call Attributes can be found in the DEVSPECIFIC portion of the LINECALLINFO structure. The Call Attribute Info is presented in the format of an array because Silent Monitoring and Call Recording could happen at the same time.

```
DWORD CallAttrtibuteInfoOffset;
DWORD CallAttrtibuteInfoSize;
DWORD CallAttrtibuteInfoElementCount;
DWORD CallAttrtibuteInfoElementFixedSize;
```

Offset pointing to array of the following structure:

```
typedef struct CallAttributeInfo{
    DWORD CallAttributeType;
    DWORD PartyDNOffset;
    DWORD PartyDNSize;
    DWORD PartyPartitionOffset;
    DWORD PartyPartitionSize;
    DWORD DeviceNameOffset;
    DWORD DeviceNameSize;
}CallAttributeInfo;
```

enum CallAttributeType

```
{
CallAttribute_Regular                    = 0,
CallAttribute_SilentMonitorCall,
CallAttribute_SilentMonitorCall_Target,
CallAttribute_RecordedCall,
CallAttribute_WhisperCoachingCall,
CallAttribute_WhisperCoachingCall_Target,
CallAttribute_Recorded_Automatic,
CallAttribute_Recorded_AppInitiatedSilent,
CallAttribute_Recorded_UserInitiatedFromApp,
CallAttribute_Recorded_UserInitiatedFromDevice
} ;
```

For recorded calls, if the application negotiates an extension less than 0x000C0000 the CallAttributeType is set to CallAttribute_RecordedCall. If the application negotiates an extension version equal to 0x000C0000 or higher, the CallAttributeType is set to CallAttribute_Recorded_Automatic, CallAttribute_Recorded_AppInitiatedSilent, CallAttribute_Recorded_UserInitiatedFromApp, or CallAttribute_Recorded_UserInitiatedFromDevice.

Call Recording Enhancement

Cisco Unified Communications Manager Release 9.0 the Call Recording feature is enhanced to allow user to start/stop current active call recording by pressing softkey on IP phone. Record key toggles between start and stop modes.

When TAPI Application invokes Recording Start / Stop APIs it has the same effect as if the user pressed the Record key on his IP phone. In addition, applications can control whether or not the phone display indication of on-going recording.

The data types that are used by Cisco TSP to report recording configuration and recording type to TAPI applications are re-worked in order to reflect the recent changes in UCM.

Interface Changes

- LineDevCaps::DevSpecific change (Cisco Extention 0x000C0000) LINEDEVCAPS, on page 248
- LineCallInfo::DevSpecific change (Cisco Extention 0x000C0000) LINECALLINFO, on page 320
- CciscoLineDevSpecificStartCallRecording Start Call Recording, on page 369
- CciscoLineDevSpecificStopCallRecording StopCall Recording, on page 370

Message Sequence

There are no changes to the message sequence with this enhancement.

Backward Compatability

This feature is backward compatible.

CallFwdAll Notification

This enhancement allows TAPI applications to distinguish off-hook calls (outgoing calls) from calls made by using the CFwdAll softkey.

TAPI provide two new additional mask bits in the existing bitmask, CallAttributeBitMask, as a part of LINECALLINFO::DEVSPECIFIC. For normal outgoing calls, the new mask is set to 0 and if a call is generated due to CFwdAll activation or deactivation, the corresponding new bit is set to 1.

Cases where the user presses CFwdAll softkey for an on-hook device are addressed, but when users go off-hook first and then press CFwdAll softkey, are not covered.

Interface Changes

In the CallAttributeBitMask field, LINECALLINFO::DEVSPECIFIC is modified to include the two new bit masks, TSPCallAttribute_CallForwardAllSet and TSPCallAttribute_CallForwardAllClear. For more information, see Details, on page 326.

Message Sequences

See CallFwdAll Notification, on page 493.

Backward Compatibility

This feature is backward compatible.

Cisco Unified TSP Auto Update

Cisco Unified TSP supports auto update functionality, so the latest plug-in can be downloaded and installed on a client machine. Be aware that the new plug-in will be QBE compatible with the connected CTIManager. When the Cisco Unified Communications Manager is upgraded to a newer version, and Cisco Unified TSP auto update functionality is enabled, the user will receive the latest compatible Cisco Unified TSP, which will work with the upgraded Cisco Unified Communications Manager. This ensures that the applications work as expected with the new release (provided the new Cisco Unified Communications Manager interface is backward compatible with the TAPI interface). The locally installed Cisco Unified TSP on the client machine allows

applications to set the auto update options as part of the Cisco Unified TSP configuration. The user can opt for updating Cisco Unified TSP in the following different ways:

- Update Cisco Unified TSP whenever a different version (higher version than the existing version) is available on the Cisco Unified Communications Manager server.
- Update Cisco Unified TSP whenever a QBE protocol version mismatch exists between the existing Cisco Unified TSP and the Cisco Unified Communications Manager version.

CIUS Session Persistency

Wireless devices introduced by Cisco such as CIUS have the capability to move between WiFi networks and also across WiFi and VPN networks (over 3G/4G) and still retain their registration with the same CiscoUCM. However, due to the change in the network the IP address of the device might undergo a change. The same scenario is applicable for docking and undocking of the CIUS phones.

To indicate this change in IP address of wireless devices such as Cius, TAPI will expose the changed IP address for lineDevices and phoneDevices through DEVCAPS structure.

TAPI will send notification to Applications on change in IP Address information on lineDevices of CIUS Device. LINE_DEVSPECIFIC notification is fired on all Open lines on a CIUS Device. TSP would fire LINE_DEVSPECIFIC event with param1 = SLDSMT_LINE_PROPERTY_CHANGED,param2 = LPCT_DEVICE_IPADDRESS.

TAPI will send notification to Applications on change in IP Address information on phoneDevices. PHONE_DEVSPECIFIC notification is fired on the phoneDevices. TSP would fire PHONE_DEVSPECIFIC event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 = PPCT_DEVICE_IPADDRESS.

TAPI will expose the changed IP address, IP Addressing Mode (IPv4, IPv6) of lineDevices in the devspecific data of LINEDEVCAPS when lineGetDevCaps API is invoked with Extension version 0x00090001 or higher.

TAPI will expose the changed IP address, IP Addressing Mode (IPv4, IPv6) of phoneDevices in the devspecific data of PHONEDEVCAPS when phoneGetDevCaps API is invoked withExtension version 0x00090001 or higher.

Note CIUS devices are SIP end points and like all other SIP end Points, they currently don't support IPV6.

Interface Changes

No interface changes.

Message Sequences

See CIUS Session Persistency, on page 545.

Backward Compatibility

This feature is backward compatible.

Click to Conference

The Click to Conference feature enables users to create conferences from an Instant Messaging (IM) application without creating a consult call first. The Cisco TSP treats the feature as an existing conference model; however, when the conference is created or dropped, the CtiExtendedReason may come as Click2Conference.

Interface Changes

None.

Message Sequences

See Click to Conference, on page 548.

Backward Compatibility

This feature is backward compatible.

CCMEncryption Enhancements

Starting with release Cisco Unified Communications Manager 10.0(1), Encryption method, which is used to encrypt the user login password, is enhanced. Older CiscoTSP clients (9.x or earlier) use Symmetric Key Encryption. Starting with rlease 10.x, the CiscoTSP client is enhanced to use a combination of Asymmetric and Symmetric Encryption mechanism. This enhancement provides more security for user credentials in non-secured connections.

Cisco recommends that applications/users upgrade Cisco TAPI clients to take advantage of this security enhancement.

To maintain backward compatibility from CTI, a new CTI Service Parameter is introduced - **Require Public key Encryption.**

The default value for this Service Parameter for this Release is **False**.

On False: CTI/CUCM allows applications/CiscoTSP clients using symmetric encryption method and Asymmetric Public key encryption method to connect with CUCM.

On True: Only CiscoTSP clients/applications which use asymmetric PublicKey Encryption Method will be able to open provider with Cisco Unified Communications Manager 10.x.

Cisco recommends that applications upgrade Cisco TAPI clients and set this service parameter to "true". The proposed plan for future releases is to set default value as true for this service parameter and later deprecate it to ensure that applications do not use older CiscoTSP Clients which use Symmetric Encryption method.

Interface Changes

There are no interface changes for this feature.

Message Sequence

CCMEncryption Enhancements, on page 544

Backward Compatibility

As mentioned above, new CTI Service parameter is added to maintain backward compatibility.

Conference Enhancements

The Conference feature of Cisco Unified Communication Manager has been enhanced with the following functions:

- Allowing a noncontroller to add another party into an ad hoc conference.

  Applications can issue the lineGetCallStatus against a CONNECTED call of a noncontroller conference participant and check the dwCallFeatures before adding another party into the conference. The application should have the PREPAREADDCONF feature in the dwCallFeatures list if the participant is allowed to add another party.

- Allowing multiple conferences to be chained.

Be aware that these features are only available if the 'Advanced Ad-hoc Conference' service parameter is enabled on the Cisco Unified Communications Manager.

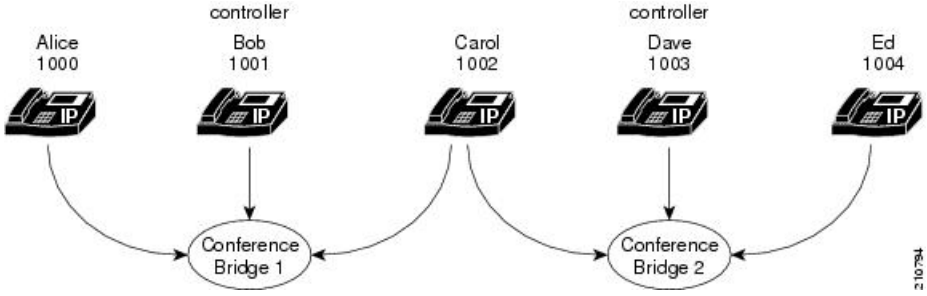
When this service parameter is changed from enabled to disabled, the system no longer allows new chaining between ad hoc conferences. However, existing chained conferences will stay intact. Any participant who is brought into the ad hoc conference by a noncontroller before this change will remain in the conference, but they can no longer add a new participant or remove an existing participant.

To avoid ad hoc conference resources remaining connected together after all real participants have left, Cisco Unified Communications Manager will disallow having more than two conference resources connected to the same ad hoc conference. However, using a star topology to connect multiple conferences could yield better voice quality than a linear topology. A new advanced service parameter, 'Non-linear Ad Hoc Conference Linking Enabled', lets an administrator select the star topology.

A participant can use the conference, transfer, or join commands to chain two conferences together. When two conferences are chained together, each participant only sees the participants from their own conference, and the chained conference appears as a participant with a unique conference bridge name. In other words, participants do not have a full view of the chained conference. The system treats the conferences as two separate conferences, even though all the participants are talking to each other.

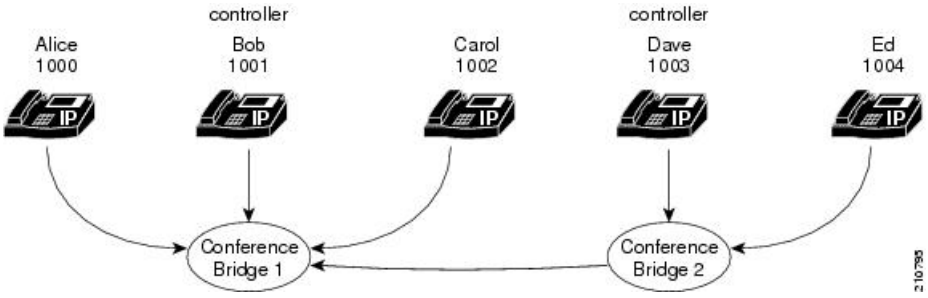
The following figures shows how TSP presents a conference model in the case of conference chaining. A, B, and C are in conference-1, and C, D, and E are in conference-2. C has an ONHOLD call on conference-1 and an active call on conference-2.

Figure 4: Conference Before Join



C then does a join with the primary call from conference-1. For A, B, and C, the conference participants comprise A, B, C, and conference-2. For D and E, the conference participants comprise D, E, and conference-1.

Figure 5: Conference After Join



When a user removes a CONFERENCE from its conference list on the phone, the operation actually drops the chained conference bridge. In the previous example, the two chained conferences have been unchained. Conference-1 will remain active and has A, B, and C as participants. However, conference-2 will become a direct call between Dave and Ed because they are the only two parties left in the conference.

Applications can achieve conference chaining by issuing a JOIN or TRANSFER on two separated conference calls. However, a LineCompleteTransfer with a conference option will fail due to a Microsoft TAPI limitation on this standard API. The application can use the Cisco LineDevSpecific extension to issue the join request to chain multiple conferences together.

Note As of Cisco Unified Communications Manager Release 8.6, Cisco TelePresence MCU conference bridges are supported through JTAPI/TSP. From a JTAPI/TSP perspective, these conference bridges behave in the same way as other supported conference bridges.

CTI Port Third-Party Monitoring Port

Opening a CTI port device in first-party mode means that either the application is terminating the media itself at the CTI port or that the application is using the Cisco Wave Drivers to terminate the media at the CTI port. This also comprises registering the CTI port device.

Opening a CTI port in third-party mode means that the application is interested in just opening the CTI port device, but it does not want to handle the media termination at the CTI port device. An example of this would be a case where an application would want to open a CTI port in third-party mode because it is interested in monitoring a CTI port device that has already been opened/registered by another application in first party mode. Opening a CTI Port in third-party mode does not prohibit the application from performing call control operations on the line or on the calls of that line.

Cisco Unified TSP allows TAPI applications to open a CTI port device in third-party mode via the lineDevSpecific API, if the application has negotiated at least extension version 6.0(1) and set the high order bit, so the extension version is set to at least 0x80050000.

The TAPI architecture lets two different TAPI applications that are running on the same PC use the same Cisco Unified TSP. In this situation, if both applications want to open the CTI port, problems could occur. Therefore, the first application to open the CTI port will control the mode in which the second application is allowed to open the CTI port. In other words, all applications that are running on the same PC, using the same

Cisco Unified TSP, must open CTI ports in the same mode. If a second application tries to open the CTI port in a different mode, the lineDevSpecific() request fails.

CTI Remote Device

This feature provides the TSP/CTI applications to extend its ability to monitor and have limited call control capability over third-party devices of a User. This capability is provided to users by representing all the third-party devices/end points of a user as a Remote Destinations configured on a virtual device type named as "CTI Remote Device".

"CTI Remote Device" is a new type of Virtual Device, can be configured from Admin pages just like any other device and need to be associated with End User with Mobility support enabled. Remote Destinations can be configured on CTI Remote Device page and each of these Remote Destinations will be configured with the Number which can be used to dial any Third Party devices (PSTN, Mobile or other PBX) and thus each of these Remote Destinations will be representing one of the third-party devices of a User which are not actually registered to Unified Communications Manager.

"CTI Remote Device" can be configured with 5 Lines, each of them shared with Enterprise Phone's Lines. On any incoming call to CTI Remote Device, the call will be extended to all the Remote Destinations/Third Party Devices configured/associated with CTI Remote Device. Call will be reported to Applications on a line on CTI Remote Device, which represents the call that is extended to Third Party Devices. Call events will be reported like other normal calls representing the state of the calls extended to Third Party Devices.

"Cisco Unified Client Services Framework" (CSF) Devices are also enhanced to be registered in Extend Mode from Jabber Clients. In the Extend Mode CSF devices behave exactly like CTI Remote Devices. Capability to configure and associate Remote Destinations to CTI Remote Device is also extended to CSF Devices.

Remote Destinations can be configured from CUCM Admin Device pages of CTI Remote Device and CSF Device or from Remote Destination pages (add new and associate it to CTI Remote/CSF device) or from TSP applications using Add/Update/Remove Remote Destination feature. The max number of remote destinations can be configured for a single CTI Remote Device depends on its owner user's max remote destinations configuration on the CUCM Admin user page (Default is 4).

Following are the supported features on CTI Remote Devices in this Release.

1. Receiving Incoming Enterprise Calls
2. Make Call (DVO -Dial via Office)
3. Call Disconnect
4. Hold / UnHold(Resume/Retrieve)
5. Redirect
6. DSS(Device State Server), and DND -(Do Not Disturb) and CPN (Globalized Calling Party)
7. Call Forwarding (Busy, Forward All ...)
8. Transfer and Direct Transfer
    1. only Direct Transfer on Same Line (DTSL) is supported
9. Conference and Join
    1. DropAnyParty Feature is also supported

2. only Join on Same Line (JSL) is supported

10. Add/Update/Remove Remote Destinations
11. URI dialing

Refer to FFS for detailed information on feature (EDCS # 1048080).

To support this feature, CiscoTSP enumerates and exposes the newly added "CTI Remote Device" and expose the Remote Destination information configured on the "CTI Remote Device" to applications. New Line Type is added for CTI Remote Device, which is exposed to Application in Devspecific part of LINEDEVCAPS (LINEDEVCAPS::DevSpecific::dwLineTypeFlags = LINEDEVCAPSDEVSPECIFIC_REMOTEDEVICE (0x00000008)) using which Remote Device Lines are identified.

It provides applications/Users capability to Add new Remote Destinations and Remove/Update existing Remote Destinations configured on "CTI Remote Device" or CSF Devices. CiscoTSP would provide applications the capability to monitor and control incoming and outgoing calls.

Note Note: This feature requires a Cisco Jabber client and this functionality is intended to be supported in Jabber for Windows 9.1

Interface Changes

The following new interfaces were added to support this feature:

- CciscoLineDevSpecificAddRemoteDestination Add Remote Destination, on page 382
- CciscoLineDevSpecificRemoveRemoteDestination Remove Remote Destination, on page 383
- CciscoLineDevSpecificUpdateRemoteDestination Update Remote Destination, on page 384

The following new error codes were added to support this feature:

Error	Description
LINEERR_DUPLICATE_INFORMATION	Reported on new LineDevSpecific Extensions (Add and Update) when the remote Destination Information which application is trying to update or add is already available.
LINEERR_REMOTE_DESTINATION_UNAVAIL	Reported on new LineDevSpecific Extensions (Update and Remove) when the RemoteDestination Number provided is not present on the Device/Line.
LINEERR_REMOTE_DESTINATION_LIMIT_EXCEEDED	Reported on new LineDevSpecifc Extension (Add) when total Remote Destination count has exceeded the limit of Remote Destinations configured for Owner User ID associated with Cti Remote Device/CSF device.
LINEERR_OPERATION_FAIL_NO_ACTIVE_RD_SET	Reported if any feature related operation request on CTI RemoteDevice failed as Active RD is not set.

Error	Description
LINEERR_ENDUSER_NOT_ASSOCIATED_WITH_DEVICE	Reported if any feature related operation request on CTI RemoteDevice failed as there is no Associated End User associated with Device.

See device specific extensions.

Message Sequences

See CTI Remote Device, on page 563.

Backward Compatibility

This feature is backward compatible.

Application Dial Rule Support

Starting with Cisco Unified Communication Manager Release 9.1 a matching Application Dial Rule is applied prior to digit analysis when a call is offered to Remote Destination associated with a CTI Remote Device. When an application adding or updating Remote Destination on a CTI Remote Device is a part of verification process the Application Dial Rules are applied before digit analysis.

Interface Change

No Interface changes

Message Sequensce

Not Applicable

Backward Compatibility

In Cisco Unified Communication Manager Release 9.0 Application Dial Rules were not applied to Remote Destinations that are associated with CTI Remote Device.

DTMF Support

Starting with Cisco Unified Communication Manager Release 9.1, applications are able to invoke lineGenerateDigits() API on a CTI Remote Device. Only out-of-band DTMF is supported in Cisco Unified Communication Manager Release 9.1.

Interface Change

No Interface changes

Message Sequensce

Not Applicable

Backward Compatibility

This is a new feature and is backward compatible.

Extend Mode Support for CSF Is Removed

In Cisco Unified Communication Manager Release 9.0 a CSF device had the ability to add Remote Destinations, starting with Cisco Unified Communication Manager Release 9.1 CSF devices can no longer register with CTI in extend mode as CTI Remote Devices and CSF devices will not be able to add Remote Destination.

Interface Change

No Interface changes

Message Sequensce

Not Applicable

Backward Compatibility

Application upgrades from Cisco Unified Communication Manager Release 9.0 to Release 9.1 or later results in the removal of Remote Destinations configured for CSF devices.

Remote Destination Reachability Verification

In Cisco Unified Communication Manager Release 9.1, an enhancement is added in CTI to verify Remote Destination reachability when it is added or updated on a CTI Remote Device. To determine if the destination is reachable, CTI performs digit analysis based on Reroute CSS configured on the CTI Remote Device. The reachability verifies that the outside dial prefix, CSS and route pattern are configured correctly.

If the destination is not reachable the request returns the error: CTIERR_EXTEND_AND_CONNECT_DESTINATION_NOT_REACHABLE.

Error code

LINEERR_REMOTE_DESTINATION_NOT_REACHABLE – can be returned when an application attempts to add or update a Remote Destination and it cannot be reached.

Interface Change

No Interface changes

Message Sequensce

Not Applicable

Backward Compatibility

Backward compatibility issues may be seen when upgrading to Cisco Unified Communication Manager Release 9.1. In Unified Communication Manager Release 9.0 destination reachability was not verified and there were no errors returned.

Persistent Connection

Persistent connection or Persistent call refers to a call between a CTI remote device and a remote destination that stay connected even when no active customer calls exist. For example: At the beginning of the day, a Cisco Unified Communications Manager server phones a teleworker at home to establish a persistent connection that remains active all day, until the application drops the call.

At least one remote destination must be configured and active on the CTI remote device in order to create a persistent call. One persistent call for each remote device is allowed at a time. No feature invocations, such as park, hold, conference, and transfer, are allowed on persistent calls.

Once a persistent call is created it remains connected until application drops the call or the maximum call duration timer expires. A persistent call is also disconnected when a remote destination drops the call or is no longer active.

Persistent calls cannot be dropped if an active call to the remote device exists. Therefore, if there is an active call, the persistent call will be dropped as soon as that call is finished.

A Persistent Connection Call differs from a typical call in that no media events are generated for the persistent call. An application may not always receive notification about a persistent call that was accepted (LNECALLSTATE_ACCEPTED). This notification may depend on the type of trunks and gateways used in a specific telephone network.

The Persistent Call feature enhances some TAPI APIs and introduces new APIs and error codes.

Note This feature is backward compatible and existing applications are not affected.

Create a Persistent Call

The TAPI lineMakeCall function is used to create persistent call.. The relevant data is provided in LINECALLPARAMS structure pointed to by the lpLineCallParams parameter. Cisco TSP ignores all other lineMakeCall parameter for a persistent call.

For a persistent call, the LINECALLPARAMS contains the following data:

- DevSpecific part refering to Cisco_CallParamsDevSpecific structure where DevSpecificFlags is set to Cisco_CALLPARAMS_DEVSPECIFICFLAGS_PERSISTENT CALL (0x00000002)
- CallingPartyID set to a directory number that appears as a remote destination CallerID directory number
- DisplayableAddress set to a name that appears as the remote destination CallerIDName.

Drop a Persistent Call

Use the standard TAPI lineDrop function to drop or disconnect persistent call. The hCall parameter should specify the persistent call handle (HCALL) returned by lineMakeCall when the persistent call was created.

Persistent Call State Change

When the persistent call status changes, an application receives a standard TAPI LINE_CALLSTATE message. For a persistent call, the new call state in the dwParamas1 field will be constructed as follows:

- The low-order 24 bits are set to one of the LINECALLSTATE_constants as they are for a regular call.

- The high-order 8 bits is set to 0x20. A corresponding bitmask is defined in CiscoLineDevspecificMsg header file as CLDSMET_PERSISTENT_CALL_STATE (0x200000000).

Persistent Call Attribute Bit Mask

A new bit definition, TSPCallAttribute_PersistentCall(0x00004000), is added to the CallAttributeBitMask enumeration in CiscoLineDevSpecificMsg.h header file. For a persistent call, a corresponding bit is turned on in the CallAttributeBitMask field in Cisco TSP extension of the TAPI LINECALLINFO structure.

Interface Changes

- lineMakeCall, on page 173 TAPI Line Functions – lineMakeCall – Note added
- LINECALLINFO, on page 320 DevSpecific change - (Cisco Extention 000D0000 – new bit definition added for Call Attribute Type - TSPCallAttribute_PersistentCall(0x00004000))
- LINECALLPARAMS, on page 336 Cisco Device-Specific Extentions – LINECALLPARAMS section added.

The following new error codes were added to support this feature:

Error	Description
LINEERR_PERSISTENT_CALL_CREATE_FAILED	Attempt to create persistent call failed.
LINEERR_PERSISTENT_CALL_ALREADY_EXISTS	Persistent call cannot be created because another one already exists.
LINEERR_OPERATION_NOT_ALLOWED_ON_PERSISTENT_CALL	No feature invocation is allowed on persistent call, such as park, hold, conference, and transfer.
LINEERR_PERSISTENT_CALL_DROP_FAILED_CALL_ACTIVE	An attempt to disconnect persistent call while customer call is still active
LINEERR_NO_PERSISTENT_CALL_EXISTS	An attempt to create announcement call on a device where persistent call does not exist
LINEERR_PERSISTENT_CALL_NOT_ESTABLISHED	An attempt to create announcement call while persistent call is still being setup
LINEERR_OPERATION_NOT_AVAILABLE_IN_CURRENT_STATE	Requested operation is not allowed in current call state

Usage Cases

See Persistent Connection Use Cases, on page 936.

Backward Compatibility

This enhancement is backward compatible and existing applications will not be affected with introduction of this enhancement.

Announcement Call

Cisco Extend and Connect is enhanced with an ability to play announcements to a remote destination. In order to play the announcement, an application creates a special type of call -Announcement Call -and identifies an announcement to be played.

The announcement can be played only to a remote destination with an existing Persistent Call. Only announcements that were uploaded to the Unified Communications Manager can be played.

Applications are notified that an announcement started by the LINE_DEVSPECIFIC event: SLDSMT_ANNOUNCEMENT_STARTED.

Applications are notified that an announcement stopped by the LINE_DEVSPECIFIC event: SLDSMT_ANNOUNCEMENT_ENDED.

Create Announcement Call

The TAPI lineMakeCall function is used to create announcement call. The relevant data is provided in LINECALLPARAMS structure pointed to by the lpLineCallParams parameter. CiscoTSP ignores all other lineMakeCall parameters in the case of announcement call.

In the case of an announcement call, the following data is provided in the LINECALLPARAMS:

- DevSpecific part refers to Cisco_CallParamsDevSpecific structure where the DevSpecificFlags is set to Cisco_CALLPARAMS_DEVSPECIFICFLAGS_ANNOUNCEMENTCALL
- CallData is set to a media content identifier (announcementID)

Drop Announcement Call

The standard TAPI lineDrop function drops or disconnects the announcement call. The hCall parameter specifies the announcement call handle (HCALL) returned by lineMakeCall when the announcement call is created.

Announcement Call State Change

When the status of the announcement call changes, an application receives the standard TAPI LINE_CALLSTATE message. For an announcement call, the construction of the new call state in the dwParams1 field is:

- The low-order 24 bits are set to one of the LINECALLSTATE_constants, which is the same as a regular call.
- The high-order 8 bits are set to 0x40. A corresponding bitmask is defined in the CiscoLineDevspecificMsg header file as follows:

  CLDSMT_ANNOUNCEMENT_CALL_STATE 0x40000000

Announcement Call Attribute Bit Mask

A new bit definition, TSPCallAttribute_AnnouncementCall (0x00008000), is added to the CallAttributeBitMask enumeration in the CiscoLineDevSpecificMsg.h header file. For an announcement call, a corresponding bit is turned on in the CallAttributeBitMask field in the Cisco TSP extension of the TAPI LINECALLINFO structure.

Interface Changes

- **Cisco_LineCallInfo_Ext000D0000extD0** is added. See LINECALLINFO, on page 320.
- Cisco_CALLPARAMS_DEVSPECIFICFLAGS_ANNOUNCEMENTCALL is added. See LINECALLPARAMS, on page 336.
- lineMakeCall:

  Beginning with Cisco Unified Communications Manager Release 10.01, applications can use lineMakeCall to create a Persistent Call or Announcement Call. For a Persistent Call or Announcement Call, the relevant data is provided in the LINECALLPARAMS structure pointed to by the lpLineCallParams parameter. All other lineMakeCall parameters are ignored in these cases.

- See Announcement Events, on page 396.

Message Sequence

See Announcement Call, on page 470.

Backward Compatibility

This enhancement is backward compatible and existing applications are not affected by the introduction of this enhancement.

NuRD (Number Matching for Remote Destination) Support

In CUCM 10.0, the existing "Cisco Extend and Connect" feature includes number matching for remote destination support. When users directly call a number that is configured as a remote destination for CTI Remote Device (CTI RD), and that remote destination is set to be active, the call is offered on the CTI Remote Device and extended to the remote destination. The called party is presented to the application as the CTI RD. If active remote destination is not set, when users call a remote destination number, a direct call between the caller and the remote destination occurs. This scenario also applies to a remote destination making a call to an enterprise directory number. If the remote destination is set to be active, from an application perspective, the CTI RD appears to initiate the call to the enterprise dn. If the active remote destination is not set, when the remote destination calls an enterprise dn, it is a direct call between the remote destination and the enterprise dn.

For those calls from and to a remote destination number, all existing features allowed on CTI RD can be performed.

Interface Changes

There are no interface changes for this feature.

Use Cases

See NuRD (Number Matching for Remote Destination) Support, on page 925.

Backward Compatibility

This feature can change the existing expected behavior in regards to calls to and from remote destination numbers directly. Applications that do not want to leverage this NuRD feature can keep the cluster-wide service parameter "Reroute Remote Destination Calls to Enterprise Number" set to false. Enabling it will enable the NuRD features. This parameter by default is set to false.

Mobility Interaction Support

The "Cisco Extend and Connect" feature includes mobility interaction. Users can specify remote destinations that are shared between the CTI Remote Device (CTI RD) and the Remote Destination Profile (RDP). When both the CTI RD and the RDP are configured for the same user, and if the application is active (active rd is set), the CTI RD processes the call first and then offers the call to the RDP. If the application is not active, the RDP processes the call first and does not offer the call to the CTI RD. When only the CTI RD is configured for a user, the existing "Cisco Extend and Connect" feature behavior with remote destinations remains unchanged. When only RDP is configured for a user, there is no application support because the devices are not controllable from a CTI.

Interface Changes

There are no interface changes for this feature.

Use Cases

There are no new use cases for this feature. (Add/Update/Delete Remote Destination) are added for CTIRD, and are applicable (without any change from App/User).

Backward Compatibility

There are no backward compatibility issues for this feature.

Call Forwarding

Starting with Cisco Unified Communications Manager 10.0(1), a new feature called “CTI Rd Call Forward” allows users to control when incoming calls are forwarded to all configured Remote Destinations on the CTI Remote Device, when no active remote destination is set.

A new check box, **Route calls to all remote destinations when client is not connected**, is added to the Cisco Unified Communications Manager device window. The check box determines whether calls are routed to all remote destinations when Active Remote Destination is not set.

When you enable the **Route calls to all remote destinations when client is not connected** check box, and Active Remote Destination is not set, the call is routed to all remote destinations. If this check box is disabled, and Active Remote Destination is not set, the call is disconnected with User_Busy error on the CTI Remote Device.

In scenarios where Active Remote Destination is set, the call is always routed to the Active Remote Destination regardless of whether the **Route calls to all remote destinations when client is not connected** check box is enabled or disabled.

Interface Changes

There are no interface changes for this feature.

Message Sequence

See CTI RD Call Forwarding, on page 641.

Backward Compatibility

There are no backward-compatibility issues for this feature.

CTI Video Support

The CTI Video Support feature allows the TAPI Application to retrieve the multimedia capabilities of Line Devices. Applications that monitor devices use this information to answer or route video calls to video capable devices.

TAPI shall expose a new structure DeviceMultiMediaCapability in Devspecific data of linedevcaps when applications issue a TSPI_LineGetDevCaps () API with Extension version 0x000D0000 or higher, on these line devices, Cisco TSP will fire **SLDSMT_LINE_PROPERTY_CHANGED** or **CPDSMT_PHONE_PROPERTY_CHANGED_EVENT** with param1= **LPCT_DEVICE_MULTIMEDIACAP_INFO** or **PPCT_DEVICE_MULTIMEDIACAP_INFO** to report Multimedia capability information change.

A new structure, DeviceCallMultiMediaCapInfo, is introduced under CiscoLineDevSpecificMsg.h, which provides the information about calling and called party multimedia capabilities that is exposed in the devspecific data of LineGetCallInfo.

Similarly, when the video capability of a calling/called of a Call changes, TSP fires **LINE_CALLDEVSPECIFIC** with param1= **SLDSMT_LINECALLINFO_DEVSPECIFICDATA** and param2= **SLDST_DEVICE_VIDEO_CAP_INFO** to report Multimedia capability information of the call

Also, devspecific data of LineCallInfo contains two new fieldsCallingPartyMultiMediaCapBitMask and CalledPartyMultiMediaCapBitMask, which indicate the fields in the DeviceMultiMediaCapInfo with valid information.

When the application makes a video call from one video enabled phone to another, the TSP fires **LINE_CALLDEVSPECIFIC** event with param1= **SLDSMT_MULTIMEDIA_STREAMSDATA** to report MultiMedia Streams information of the call. The Multimedia Streams information of the call is exposed in the Devspecific data of linecallinfo (As a part of VideoStreamInfo structure) when application issues TSPI_LineGetCallInfo() API with Extension version 0x000D0000 or higher.

The following table describes the video capabilities provided by TAPI for currently supported devices.

Device	Supports Initial Device Multimedia Capability	Supports Dynamic Video Capability Change	Reports calling and called Multimedia Capabilities on call info (LineGetCallInfo)	Supports Multimedia Streams Information
8945 (SIP)	Yes	Yes	Yes	Yes
8945 (SCCP)	Yes	Yes	Yes	No
9951, 9971(SIP)	Yes	Yes	Yes	Yes
EX60/90 (SIP)	Yes	N/A	Yes	Yes
CTS 500-32 (SIP)	Yes	N/A	Yes	Yes
Jabber(CSF/softphone mode) (SIP)	Yes	Yes	Yes	No

Device	Supports Initial Device Multimedia Capability	Supports Dynamic Video Capability Change	Reports calling and called Multimedia Capabilities on call info (LineGetCallInfo)	Supports Multimedia Streams Information
CTI RoutePoint (SCCP)	N/A	N/A	Yes	No
CTI Port (SCCP)	N/A	N/A	Yes	No
All other phones	N/A	N/A	Yes	No

Supported Features (With in the same cluster):

- Originating Call and Consult Call
- Redirect
- Call Forward
- Hold and Resume
- Hunt List
- Transfer
- Extension Mobility
- Super Provider

Supported Features (Across the cluster):

- Originating Call and Consult Call
- Redirect
- Call Forward
- Hold and Resume
- Hunt List
- Extension Mobility
- Super Provider

Limitations:

- Remote In Use
    - CiscoTSP does not provide correct calling and called party multimedia capabilities on a call that is in inactive state or is in Remote In Use state.
- MultiMedia Capability
    - Calling and called party multimedia capabilities are UKNOWN on the calling side until the called party answers the call.
    - When a call is initiated over SIP trunk configured with early offer, the called party video capabilities are the negotiated capabilities that get reported instead of the actual capability, on the called party.
    - Only video capability information is known for calls over the H323 trunk, Screen count and telepresence interop information is unknown.
- MultiMediaStreams
    - CiscoTSP does not provide multimedia streams information if the device is a SCCP phone. The CiscoTSP does not deliver SLDSMT_MULTIMEDIA_STREAMSDATA, and the

TSPI_LineGetCallInfo() API does not provide multimedia streams information in the VideoStreamInfo structure.

- Change in called party
  - In scenarios such as Shared Lines or redirect, where the called party changes, the application is notified of the new called party capability only if the called party is configured with unique display names.

Interface Changes

- LineDevCaps::DevSpecific change - (Cisco Extention 000D0000) LINEDEVCAPS, on page 317
- LineCallInfo::DevSpecific change - (Cisco Extention 000D0000) LINECALLINFO, on page 320
-
- LPCT_DEVICE_MULTIMEDIACAP_INFO – Indicates or notifies application that Device Multi Media Capability Information on the Line/Device has changed. Line Property Changed Events, on page 404
- PPCT_DEVICE_MULTIMEDIACAP_INFO – Indicates or notifies application that Device Multi Media Capability Information on the Line/Device has changed. Phone Property Changed Events, on page 405
- SLDST_DEVICE_VIDEO_CAP_INFO – (New bit mask type added for Param2 bits on SLDSMT_LINECALLINFO_DEVSPECIFICDATA) LINECALLINFO_DEVSPECIFICDATA Events, on page 402
- SLDSMT_MULTIMEDIA_STREAMSDATA – (New Message Type in Line_DevSpecific Message) MultiMedia Streams Data Notification Event, on page 409

Message Sequences

See Video Capabilities and Multimedia Information, on page 642.

Backward Compatibility

This feature is not backward compatible.

Device State Server

The Device State Server feature provides accumulative state of all the lines on the device. Applications are notified about the device status through the PHONE_DEVSPECIFIC and LINE_DEVSPECIFIC events.

An application for enabling the Device State Server support needs to set the DEVSPECIFIC_DEVICE_STATE and DEVSPECIFIC_DEVICE_STATE_STATUS_ message flags using the lineDevSpecific SLDST_SET_STATUS_MESSAGES request and the PhoneDevSpecific CPDST_SET_DEVICE_STATUS_MESSAGES request respectively.

When Cisco TSP receives the DEVICE_STATE events from CTI, it notifies the application about the accumulative state of all the lines on the device using the PHONE_DEVSPECIFIC and LINE_DEVSPECIFIC events.

The device status in the LINE_DEVSPECIFIC and PHONE_DEVSPECIFIC events can be one of the following.

```
enum lineDeviceState{
lineDeviceState_UNKNOWN = 0,
lineDeviceState_ACTIVE = 1,
lineDeviceState_ALERTING = 2,
```

```
lineDeviceState_HELD = 3,
lineDeviceState_WHISPER = 4,
lineDeviceState_IDLE = 5
};

enum PhoneDeviceState{
     PhoneDeviceState_UNKNOWN  = 0,
     PhoneDeviceState_ACTIVE = 1,
     PhoneDeviceState_ALERTING = 2,
     PhoneDeviceState_HELD = 3,
     PhoneDeviceState_WHISPER = 4,
     PhoneDeviceState_IDLE = 5
};
```

This feature provides the accumulative state of all the lines on the device or the phone to the application. Events are provided based on the following criteria:

- IDLE

  If all the lines on the device are IDLE, the device state is considered IDLE and the corresponding event is delivered to the qualified applications.

- ACTIVE

  If any of the lines on the device have an ACTIVE (call states are LINECALLSTATE_DIALTONE, LINECALLSTATE_DIALING, LINECALLSTATE_PROCEEDING, LINECALLSTATE_RINGBACK, and LINECALLSTATE_DISCONNECTED) call, the device state is considered ACTIVE and the corresponding event is delivered to the qualified applications.

- ALERTING

  If there is no ACTIVE call on any of the lines of the device and at least one of the lines has an ALERTING (call states are LINECALLSTATE_OFFERING and LINECALLSTATE_ACCEPTED) call, the device state is considered ALERTING and the corresponding event is delivered to the qualified applications.

- HELD

  If there is no ACTIVE or ALERTING call on any of the lines of the device and at least one of the lines has a HELD call, the device state is considered HELD and the corresponding event is delivered to the qualified applications.

- WHISPER

  If there is no ACTIVE or ALERTING or HELD call on any of the lines of the device and at least one of the lines have an intercom call, the device state is considered WHISPER and the corresponding event is delivered to the qualified applications.

Note The ACTIVE state has priority over the ALERTING, HELD, and IDLE states.

The ALERTING state has priority over the HELD and IDLE states.

The HELD state has priority over the IDLE state.

Note To make the LineDevSpecific event indicate the device state for any line of that device, the DEVSPECIFIC_DEVICE_STATE_STATUS_message flag for that line must be turned on using the lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

Direct Transfer

In Cisco Unified Communications Manager, the Direct Transfer softkey lets users transfer the other end of one established call to the other end of another established call, while dropping the feature initiator from those two calls. Here, an established call refers to a call that is either in the on hold state or in the connected state. The “Direct Transfer” feature does not initiate a consultation call and does not put the active call on hold.

A TAPI application can invoke the “Direct Transfer” feature by using the TAPI lineCompleteTransfer() function on two calls that are already in the established state. This also means that the two calls do not have to be set up initially by using the lineSetupTransfer() function.

Direct Transfer Across Lines

The Direct Transfer Across Lines feature allows the application to directly transfer calls across the lines that are configured on the device. The application monitors both the lines when directly transferring the calls across the lines.

A new LineDevSpecific extension, CciscoLineDevSpecificDirectTransfer, is added to direct transfer calls across the lines or on the same line. The 0x00090000 extension must be negotiated to use CciscoLineDevSpecificDirectTransfer.

Interface Changes

See Direct Transfer, on page 373.

Message Sequences

See CTI Remote Device, on page 563.

Backward Compatibility

This feature is backward compatible.

Directory Change Notification

The Cisco Unified TSP sends notification events when a device has been added to or removed from the user-controlled device list in the directory. Cisco Unified TSP sends events when the user is deleted from Cisco Unified Communications Manager Administration.

Cisco Unified TSP sends a LINE_CREATE or PHONE_CREATE message when a device is added to a users control list.

It sends a LINE_REMOVE or PHONE_REMOVE message when a device is removed from the user controlled list or the device is removed from database.

When the system administrator deletes the current user, Cisco Unified TSP generates a LINE_CLOSE and PHONE_CLOSE message for each open line and open phone. After it does this, it sends a LINE_REMOVE and PHONE_REMOVE message for all lines and phones.

Note Cisco Unified TSP generates PHONE_REMOVE / PHONE_CREATE messages only if the application called the phoneInitialize function earlier.

The system generates a change notification if the device is added to or removed from the user by using Cisco Unified Communications Manager Administration or the Bulk Administration Tool (BAT).

If you program against the LDAP directory, change notification does not generate.

Do Not Disturb

The Do Not Disturb (DND) feature lets phone users go into a Do Not Disturb state on the phone when they are away from their phone or simply do not want to answer incoming calls. The phone softkey DND enables and disables this feature.

From theCisco Unified Communications Manager user windows, users can select the DND option DNR (Do Not Ring).

Cisco TSP makes the following phone device settings available for DND functionality:

- DND Option: None/Ringer off
- DND Incoming Call Alert: Beep only/flash only/disable
- DND Timer: a value between 0-120 minutes. It specifies a period in minutes to remind the user that DND is active.
- DND enable and disable

Cisco TSP includes DND feature support for TAPI applications that negotiate at least extension version 8.0 (0x00080000).

Applications can only enable or disable the DND feature on a device. Cisco TSP allows TAPI applications to enable or disable the DND feature via the lineDevSpecificFeature API.

Cisco TSP notifies applications via the LINE_DEVSPECIFICFEATURE message about changes in the DND configuration or status. To receive change notifications, an application must enable the DEVSPECIFIC_DONOTDISTURB_CHANGED message flag with a lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

This feature applies to phones and CTI ports. It does not apply to route points.

Do Not Disturb-Reject

Do Not Disturb (DND) enhancements support the rejection of a call. The enhancement Do Not Disturb–Reject (DND–R) enables the user to reject any calls when necessary. Prior to the Cisco Unified Communications

Manager Release 7.0(1), DND was available only with the Ringer Off option. If DND was set, the call would still get presented but without ringing the phone.

To enable DND–R, access the Cisco Unified Communications Manager Administration phone page or the user can enable it on the phone.

However, if the call has an emergency priority set, the incoming call is presented on the phone even if the DND–R option is selected. This will make sure that emergency calls are not missed.

Feature priority is introduced and defined in the enum type for making calls or redirecting existing calls. The priority is defined as:

```
enum CiscoDoNotDisturbFeaturePriority {
CallPriority_NORMAL = 1
CallPriority_URGENT = 2
CallPriority_EMERGENCY = 3
};
```

Feature priority introduces LineMakeCall as part of DevSpecific data. Currently the following structure is supported in DevSpecific data for LineMakeCall:

```
typedef struct LineParams {
 DWORD FeaturePriority;
} LINE_PARAMS;
```

The new Cisco LineDevSpecific extension, CciscoLineRedirectWithFeaturePriority with type SLDST_REDIRECT_WITH_FEATURE_PRIORITY, supports redirected calls with feature priority.

Also in a shared line scenario, if one of the lines is DND–R enabled and if the Remote In Use is true, then it will be marked as connected inactive.

Interface Changes

See lineMakeCall, on page 173 and Redirect with Feature Priority, on page 367.

Message Sequences

See Do Not Disturb-Reject, on page 682.

Backward Compatibility

This feature is backward compatible.

Drop-Any-Party

The Drop-Any-Party feature enables the application to drop any call from the ad-hoc conference. This feature is currently supported from the phone interface. The application uses the LineRemoveFromConference function to drop the call from a conference. When the call is dropped from a conference, TSP receives CtiDropConferee as the call state change cause, and this is sent to TAPI as the default cause.

Interface Changes

See lineRemoveFromConference, on page 182.

Message Sequences

See Drop Any Party, on page 684.

Backward Compatibility

This feature is backward compatible. The 0x00090000 extension is added to maintain backward compatibility.

Early Offer

The Early Offer feature allows the SIP trunk to support early offer outbound calls without using MTP when the media capabilities and media port information of the calling endpoint is available. For the endpoints where the media port information is not available (for example, H323 slow start calls or delayed offer SIP calls or legacy SCCP phones) for Early Offer, Cisco Unified Communications Manager allocates an MTP to provide an offer. This means Unified CM allocates MTP only when needed.

To support the Early Offer feature, Cisco TSP introduces CCiscoLineDevSpecific extension (CciscoLineDevSpecificEnableFeatureSupport) to allow the application to enable or disable the Early Offer feature. This DevSpecific type is generic and can be used for supporting features added in the future.

The registration of CTI ports or route points are as follows:

- Dynamic registration of CTI ports or route points with Early Offer Support:
  - New LineDevSpecific type must be requested before registration.
- Static registration of CTI ports with Early Offer Support:
  - New LineDevSpecific type is requested before registration and used to change the Device Capability of Early Offer Support after registration.

GET IP and PORT EVENT reports to a CTI Port or Route Point registered with Early Offer Support enabled, on an outbound call. When an outbound call is routed through the SIP trunk with Early Offer Support, TSP reports LINE_DEVSPECIFIC event with Param1 = SLDSMT_RTP_GET_IP_PORT and Param2 = IPAddressing Mode along with SetRTP bit information (ninth bit from LSB).

dwParam2 = 0x00000xyy, where:

- x (ninth Bit from LSB) — SetRTPInfo (1 — Applications must set the RTP information and 0 — Applications must not set the RTP Information
- yy (8 bits) — IPAddressing Mode.

For dynamically-registered CTI ports or route points with Early Offer: For this notification, applications have to set the RTP information using the Existing LineDevspecific Type (CciscoLineDevSpecificSetRTPParamsForCall) with the IP and Port Information for the IPAddressing Mode reported. Applications must not set the RTP information on the Open Logical Channel notification if the application has already set the information on GetIP and Port notification (SLDSMT_RTP_GET_IP_PORT).

For statically-registered CTI ports: For this notification, applications must open and reserve the port used for registration.

To receive the Get IP and Port notification (SLDSMT_RTP_GET_IP_PORT), an application must set the DEVSPECIFIC_GET_IP_PORT message flag by using the lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

TAPI provides setRTP information in dwParam2 of OpenLogical Channel notification (LINE_DEVSPECIFIC Event with dwParam1 = SLDSMT_OPEN_LOGICAL_CHANNEL) along with the IP addressing capability using which the application must determine whether it has to set the RTP information.

dwParam2 = 0x0000xxyy, where:

- xx — SetRTPInfo (1 — Applications must set the RTP information and 0 — Applications must not set RTP information)
- yy — IPAddressing Capability.

TSP Reports New Error Code (LINEERR_REGISTER_GETPORT_SUPPORT_MISMATCH) when application tries to dynamically or statically register CTI port or route point without Early Offer Support, where as the CTI port is already Registered Dynamically/Statically with Early Offer support by other applications.

Media Driver Support for Early Offer

For an Early Offer call on a CTI Port registered with Early Offer support, when the other party has the IP and Port information of the calling party, the other party starts transmitting the media early even before the Media Events are reported on the CTI Port (registered with Early Offer).

Due to this Early Media or delay in reporting Media Reception Event to the application, Wave Driver misses initial data transmitted as the current Wave Drivers (both Legacy and Cisco New Wave Driver) supports opening of the Ports and starts reception of data only after Media Events are reported to the application.

To capture the early transmitted data, the receiving port needs to be opened after GET_IP_PORT Event and Buffer Incoming Data. Current supported APIs for opening a port in New Wave Driver and Legacy Wave Driver requires CODEC and other supported information (DSCP, SRTP Information, and Silence Type). But in the Early Offer case, this information is not available at GET_IP_PORT event and needs to add new API's to open the Port, start buffering, and then Update the endpoint with other media endpoint related information (CODEC, DSCP, SRTP Information, and Silence Type).

The following API's have been added to the New Wave Driver to capture Early Media for Early Offer Call:

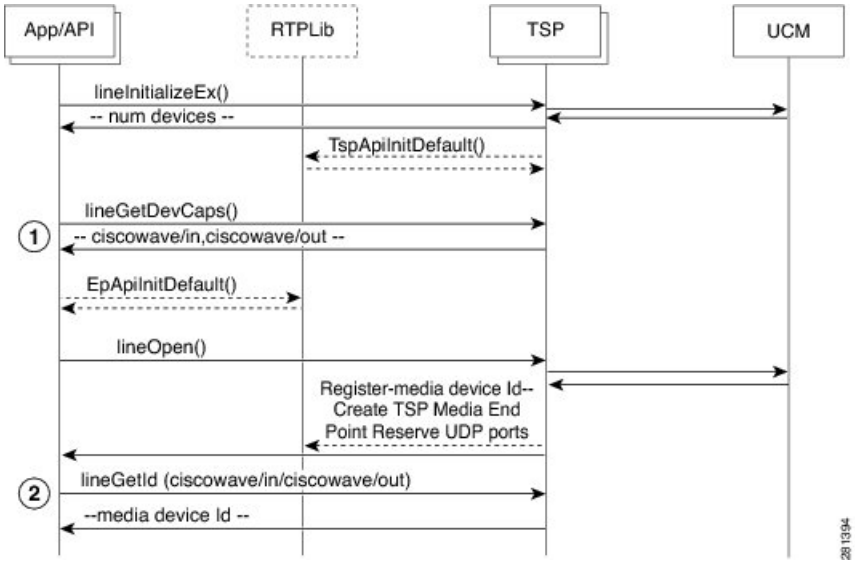
- EpStreamOpen()—Opens the Port and Starts Buffering Incoming Data.
- EpUpdateById()—Updates the Media endpoint Data Information (CODEC, DSCP, SRTP Information, and Silence Type).
  - Returns True on successful updation or false on failure; specific error code can be retrieved by calling EpApiGetLastError.
  - When the Stream is already started using EpStreamStart() API, EpUpdateById() request fails with error EP_ERR_TOAPP_INVALID_STATE.
  - When the Port is opened using EpStreamOpen() API, EpUpdateByID() will update the data information related to the media endpoint except the address and port.
  - On a Stream that is opened; In case of mismatch of LocalAddrInfo with the actual port used for Opening Socket, the request fails with error EP_ERR_ADDR_MISMATCH.

Note Applications must use these newly added APIs to capture Early Media Data for Early Offer Call.

TAPI Application Message Flow for Early Offer Call

The message flow in the following figure is described in steps 1 and 2.

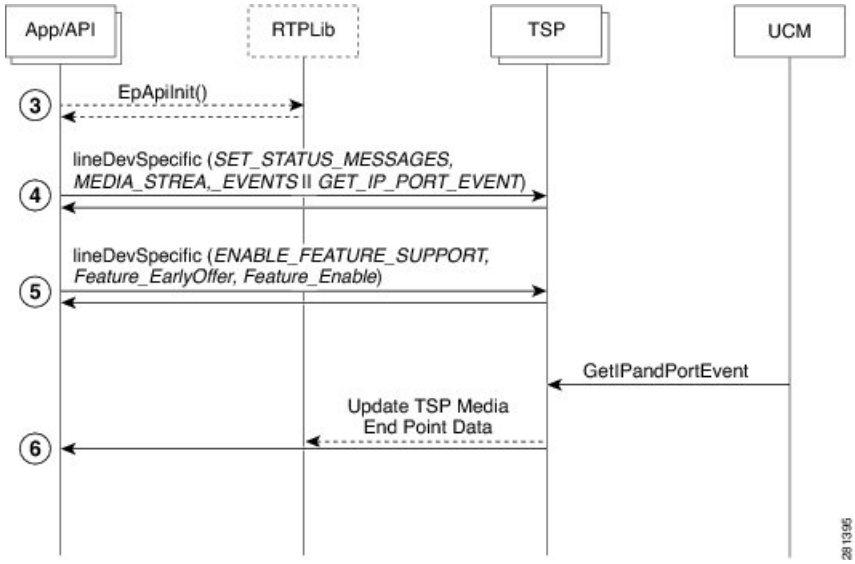
Figure 7: Application Message Flow for Early Offer Call — Steps 1 and 2



1. Initialize TAPI, get LINEINFO for the available line devices, and find the devices that are capable of using the Cisco RTP Library functionalities.
2. Get the media device identifier associated with a particular line device.

   The message flow in the following figure is described in steps 3 to 6.

   Figure 8: Application Message Flow for Early Offer Call — Steps 3 to 6

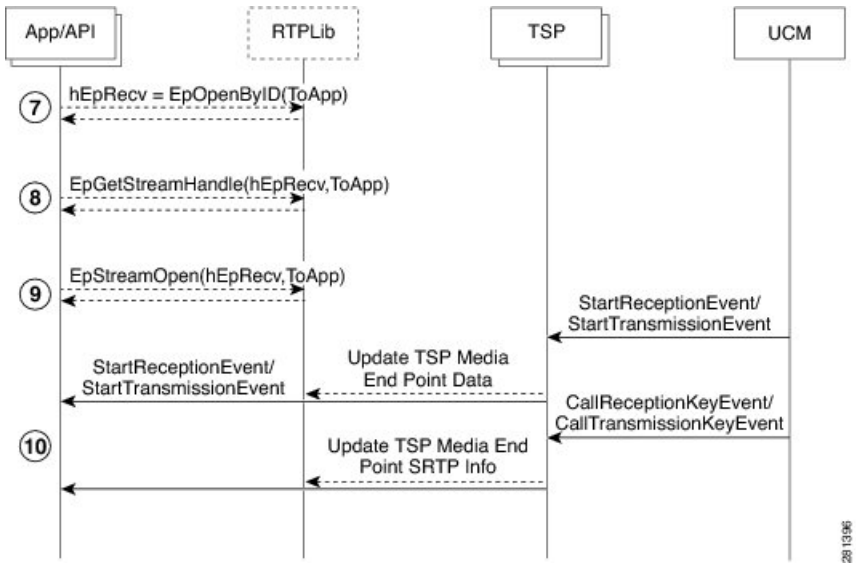


3. Initialize RTP Library.

4. Subscribe for media stream, and GetIP and Port events for the relevant devices using the Cisco lineDevSpecific extension (CciscoLineDevSpecificSetStatusMsgs).
5. Enable the Early Offer feature support on that line/device using the lineDevSpecific extension (CciscoLineDevSpecificEnableFeatureSupport).
6. GetIP and Port events reported to the application, and reports for the Early Offer call.

   The message flow in the following figure is described in steps 7 to 8.

Figure 9: Application Message Flow for Early Offer Call — Steps 7 to 10

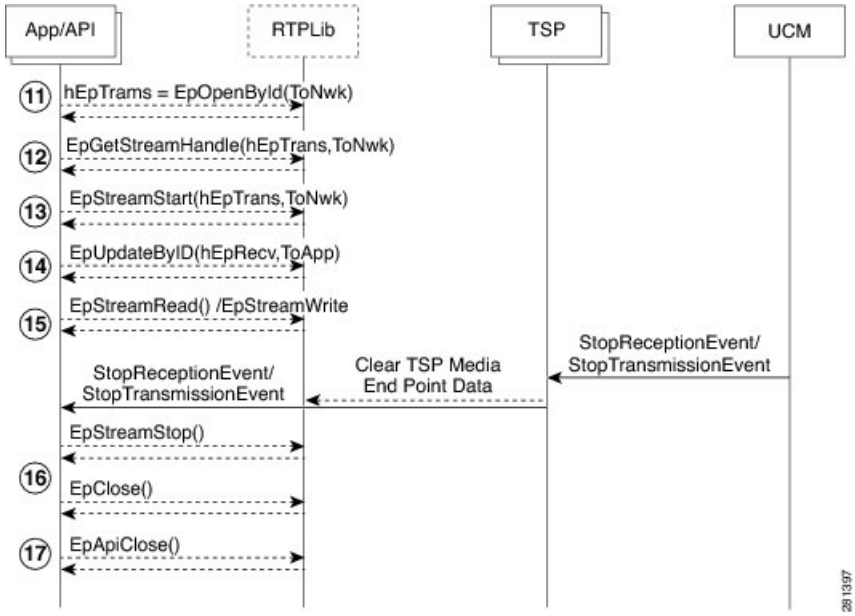


7. Create Media End Point for receiving the data.
8. Get stream handle for Media End Point created for receiving the data.
9. Open the port and start buffering the data on the receiving port.
10. Start monitoring Media Events.

    *** hEpRecv = StreamHandle for Receiving Stream

    The message flow in the following figure is described in steps 11 to 17.

Figure 10: Application Message Flow for Early Offer Call — Steps 11 to 17



11. Create Media End Point for transmitting the data.
12. Get out stream handle.
13. Start data streaming.
14. Update the opened Media End Point with CODEC and other information available in Media Event.
15. Receive/transmit data.
16. Stop data streaming and close end point.
17. Close EpAPI before exiting program.

    *** hEpTrans = StreamHandle for Transmitting Stream

Note

- For the IPV6 Registered port, GetPort Notification is not reported to the application. For the Dual Mode CTI Port or Route Point the Early Offer is supported with IPV4 addresses capability.
- For the Statically Registered CTI port with Legacy Wave driver, the Early Offer feature is not supported and TSP reports error if new LineDevspecific is requested to enable Feature Support.
- For the Statically Registered CTI port with New Cisco Wave Driver/User control Registered CTI Port, New Get IP and Port Notification applications has to open the port that is assigned for the CTI Port.
- On an Early Offer support registered CTI Port, the applications must send the RTP information about new notification and must not set the RTP information on OpenLogical Channel Notification. If set, it is failed by CTI and an acknowledgement is reported to the application.
- When IPv6 support is added, the application receives GetPort Notification twice (one for Ipv4 and one for Ipv6 address) for dual mode device. When a call is answered, the application can close the unused port based on IPAddressingType in Open Logical Channel Notification.

Note To support this feature, the application must negotiate line extension 0x000B0000 or above.

Interface Changes

See Early Offer, on page 378, Enable Feature, on page 379, Get IP and Port Event, on page 408, Set Status Messages, on page 349, and Open Logical Channel Events, on page 400.

Message Sequences

See Early Offer, on page 698.

Backward Compatibility

This feature is backward compatible.

End-to-End Call Trace

End-to-End Call Trace allows tracing of calls that traverse multiple Cisco voice products, such as Cisco Unified Communications Manager, Cisco IOS Gateway, and Cisco Call Center products.

A new tool called System Call Tracing tool is developed to collect call records from all voice platforms to trace specific calls and to troubleshoot call failure or other issues.

This tool uses the calling party number, called party number, and time stamp to find at least one call record from any voice product that it can access. From that single call record, System Call Tracing tool traces the call on all the voice products it has traversed.

Interface Changes

LINECALLINFO::DEVSPECIFIC is modified to include TSP Unique Call Reference ID. For more information, see LINECALLINFO, on page 320.

Message Sequences

See End-To-End Call Trace, on page 711.

Backward Compatibility

New extension 0x000A0000 is added to maintain backward compatibility.

EnergyWise DeepSleep Mode Support

This feature allows the phone to participate in an EnergyWise enabled system. The phone reports its power usage to a EnergyWise compliant switch to allow the tracking and control of power within the customer premise. The phone provides alternate reduced power modes including an extremely low, off mode. The Cisco Unified Communications Manager administrator configures and exclusively manages the phones power state through vendor specific configuration on the Cisco Unified CM Admin pages.

When the phone turns off power after negotiation with an EnergyWise switch, it unregisters from Cisco Unified CM and enters Deep Sleep/PowerSavePlus mode. Phones automatically re-register back with the Cisco Unified CM once the Deep Sleep mode configured PowerON time is reached.

However, for Cisco Unified IP Phones Series 9900 and 6900 phones, press the select key on the phone to wake up the phone from the Deep Sleep/PowerSavePlus mode, but there is no way to register Cisco Unified IP Phones 7900 Series phones back to the Cisco Unified CM during Deep Sleep. This is the limitation for the Cisco Unified IP Phones 7900 Series phones. You can configure Deep Sleep mode on the Device page of the Cisco Unified CM. Configure Deep Sleep mode for the phones at least 10 minutes before the actual power off time to allow the information to synchronize between the switch and the phone.

Power off idle timer enables only in the case when there is physical interaction on the phone. For example if there is a call on the EnergyWise configured phone during the deep sleep time and the user tries to disconnect the call from the application, then the power off idle timer defaults to 10 minutes but if the user disconnects the call manually from the phone, then the power off idle timer takes the value configured on the Cisco Unified CM device page.

TAPI provides the PHONE_STATE message with dwparam1 = PHONESTATE_SUSPEND and EnergyWisePowerSavePlus reason in dwParam2 when the phone unregisters as it enters DeepSleep, and if the phone successfully negotiates with the appropriate extension version 0x000B0000 or higher.

TAPI provides the LINE_LINEDEVSTATE message with dwparam1 = LINEDEVSTATE_OUTOFSERVICE and EnergyWisePowerSavePlus reason in dwparam2 when the phone unregisters as it enters DeepSleep, and if the phone successfully negotiates with the appropriate extension version 0x000B0000 or higher.

As part of this feature TAPI exposes all out of service reason codes in the PHONESTATE_SUSPEND and LINEDEVSTATE_OUTOFSERVICE in dwParam2 when the phone unregisters, and if the phone successfully negotiates with the appropriate extension version 0x000B0000 or higher.

TAPI defines a new enum CiscoLineDevStateOutOfServiceReason in CiscoLineDevSpecificMsg.h

And enum CiscoPhoneStateOutOfServiceReason in CiscoPhoneDevSpecificMsg.h

Interface Changes

```
New Enum under CiscoLineDevSpecificMsg.h
 enum CiscoLineDevStateOutOfServiceReason
{
    CiscoLineDevStateOutOfServiceReason_Unknown = 0x00000000,
    CiscoLineDevStateOutOfServiceReason_CallManagerFailure = 0x00000001,
    CiscoLineDevStateOutOfServiceReason_ReHomeToHigherPriorityCM = 0x00000002,
    CiscoLineDevStateOutOfServiceReason_NoCallManagerAvailable = 0x00000003,
    CiscoLineDevStateOutOfServiceReason_DeviceFailure = 0x00000004,
    CiscoLineDevStateOutOfServiceReason_DeviceUnregistered = 0x00000005,
    CiscoLineDevStateOutOfServiceReason_EnergyWisePowerSavePlus = 0x00000006,
    CiscoLineDevStateOutOfServiceReason_CtiLinkFailure = 0x00000101
};

New Enum under CiscoPhoneDevSpecificMsg.h
enum CiscoPhoneStateOutOfServiceReason
{
    CiscoPhoneStateOutOfServiceReason_Unknown = 0x00000000,
    CiscoPhoneStateOutOfServiceReason_CallManagerFailure = 0x00000001,
    CiscoPhoneStateOutOfServiceReason_ReHomeToHigherPriorityCM = 0x00000002,
    CiscoPhoneStateOutOfServiceReason_NoCallManagerAvailable = 0x00000003,
    CiscoPhoneStateOutOfServiceReason_DeviceFailure = 0x00000004,
    CiscoPhoneStateOutOfServiceReason_DeviceUnregistered = 0x00000005,
    CiscoPhoneStateOutOfServiceReason_EnergyWisePowerSavePlus = 0x00000006,
```

```
    CiscoPhoneStateOutOfServiceReason_CtiLinkFailure = 0x00000101
};
```

Message Sequences

See EnergyWise Deep Sleep Mode Use Cases, on page 744

Backwards Compatibility

This feature is backward compatible.

Extension Mobility

Extension Mobility, a Cisco Unified Communications Manager feature, allows a user to log in and log out of a phone. Cisco Extension Mobility loads a user Device Profile (including line, speed dial numbers, and so on) onto the phone when the user logs in.

Cisco Unified TSP recognizes a user who is logged into a device as the Cisco Unified TSP User.

Using Cisco Unified Communications Manager Administration, you can associate a list of controlled devices with a user.

When the Cisco Unified TSP user logs into the device, the system places the lines that are listed in the user Cisco Extension Mobility profile on the phone device and removes lines that were previously on the phone. If the device is not in the controlled device list for the Cisco Unified TSP User, the application receives a PHONE_CREATE or LINE_CREATE message. If the device is in the controlled list, the application receives a LINE_CREATE message for the added line and a LINE_REMOVE message for the removed line.

When the user logs out, the original lines get restored. For a non-controlled device, the application perceives a PHONE_REMOVE or LINE_REMOVE message. For a controlled device, it perceives a LINE_CREATE message for an added line and a LINE_REMOVE message for a removed line.

Extension Mobility Cross Cluster

Extension Mobility Cross Cluster allows users provisioned in one cluster to log in to an IP phone in another cluster.

For this feature, Extension Mobility profile can be added to the control list in addition to the devices. When this profile is added to the control list and an Extension Mobility Cross Cluster user logs in to or logs out of a device within a cluster or either across the cluster, Cisco TSP notifies the application with required Phone Create/Line_Create and Phone_Remove/Line_Remove events.

Interface Changes

None

Message Sequences

See Extension Mobility Cross Cluster, on page 755.

Backward Compatibility

This feature is backward compatible.

Extension Mobility Memory Optimization Option

The Extension Mobility (EM) feature supports Cisco Unified TSP to use TAPI LINE_CREATE / LINE_REMOVE mechanism to dynamically create and remove line devices resulting from EM login or logout. TAPI, by design, does not remove a device dynamically and marks the device as 'not available'. It remains in memory until the provider is shutdown. As a result, when the EM feature is used, memory utilization grows over time (during login/logout operations) until the memory is exhausted. In many cases, the only workaround is to restart the telephony service or reboot a TAPI client machine.

The EM Memory Optimization Option feature is intended to minimize the usage of LINE_CREATE / LINE_REMOVE in EM-related scenarios by reusing TAPI device IDs for lines from different EM profiles loaded on the same IP Phone. Lines with the same index in different EM profiles share the same TAPI line device ID.

The feature can be enabled or disabled by using the registry settings. By default, the feature is disabled so that the existing applications are not affected.

If the feature is enabled, LINE_CREATE messages are used by Cisco Unified TSP only for the first time when an EM profile is loaded on a particular IP Phone. After the EM line is created, it is not removed with LINE_REMOVE. It is instead placed in the Inactive state when EM logout occurs. Operations cannot be performed when a device is in the Inactive state and the LINEERR_DEVICE_INACTIVE error returns if an operation is invoked.

The line is reactivated when an EM profile is reloaded on the IP Phone as a result of a new EM login. Along with the line reactivation notification, the application is also notified that line device capabilities have changed. The Other-Device State Notification feature is utilized for delivering active, inactive, and capability change messages to an application. For more information, see Other-Device State Notification, on page 77.

Note The EM Memory Optimization Option feature intends to minimize the usage of LINE_CREATE/LINE_REMOVE messages. Even if the feature is enabled, an application can still receive the LINE_CREATE and LINE_REMOVE messages in some scenarios. So the application has to be written in a way that it can handle the existing LINE_CREATE/LINE_REMOVE events along with the new LineActive/LineInactive notifications.

Installation/Configuration Change

This feature can be enabled or disabled using the registry settings. The registry settings are stored in the EMOptions registry entry that is created during the Cisco TSP installation or upgrade. There is only one EMOptions entry in the Cisco TSP registry settings which applies to all Cisco TSP instances on the box.

The EMOptions registry settings must be changed manually. There is no Cisco TSP configuration interface to modify the setting and no possibility to change the feature behavior dynamically. Cisco TSP must be restarted for the modified setting to take effect.

A corresponding parameter is also available for a silent Cisco TSP install. This allows enabling or disabling of the feature at the time of the silent install/upgrade process.

Interface Changes

CiscoLineDevStateCloseReason provides details to the LINEDEVSTATE_CLOSE state and is passed to the application as dwParam2 in the LINE_LINEDEVSTATE message.

```
enum CiscoLineDevStateCloseReason
{
CiscoLineDevStateCloseReason_Unknown = 0,
CiscoLineDevStateCloseReason_LineNotAvailable,
CiscoLineDevStateCloseReason_EMActivity
};
```

Message Sequences

See Extension Mobility Memory Optimization Option, on page 762.

Backward Compatibility

This feature can be enabled or disabled using the registry settings. By default, this feature has been disabled so that the existing applications are not affected.

External Call Control

External Call Control enables Cisco Unified Communications Manager to route calls based on enterprise policies and presence-based routing rules of individual users. When External Call Control is enabled, Cisco Unified Communications Manager queries the designated web services hosting the enterprise policies or user rules and routes the calls based on the routing decisions returned.

As TSP receives the expected unmodified Directory Number or partition in all of the party fields, most scenarios remain unaffected by the External Call Control feature. TSP passes these unmodified fields in the PartyId fields to TAPI. Since it is also possible for the External Call Control feature to change the modified calling and called parties, TSP passes this in the existing modified fields of the devSpecific part of lineCallInfo.

With the changes made by CTI for the External Call Control feature, TSP also supports Translation Patterns. Calls going through translation patterns are supported by the TSP and if these calls are involved in a conference, the correct number of CONFERENCE calls shall be created and maintain for the duration of the conference.

Interface Changes

New CtiReasonExternalCallControl (42) in the ExtendedCallReason field in the devSpecific part of lineCallInfo for some intercept scenarios.

New dev specific error, LINEERR_OPERATION_FAIL_CHAPERONE_DEVICE, is returned in LINE_REPLY for any request that is rejected for a device which is involved in a chaperone call except for lineSetupConference/lineAddToConference, lineDevSpecific(SLDST_START_CALL_RECORDING), and lineDrop requests.

In the CallAttributeBitMask field, LINECALLINFO::DEVSPECIFIC is modified to include a new bit mask, TSPCallAttribute_ChaperoneCall. For more information, see Details, on page 326.

Message Sequences

See External Call Control, on page 766.

Backward Compatibility

This feature is backward compatible.

Note As TSP did not support Translation Patterns before this release, the support of Translation Patterns is considered backward compatible even though there is a change in the CallInfo for calls using Translation Patterns.

FIPS Compliance

Federal Information Processing Standards (FIPS) are publicly announced standards developed by the United States federal government for use in computer systems by all non-military government agencies and government contractors. The FIPS 140-2 requirements have been defined jointly by the American NIST (National Institute for Standards and Technology) and the Canadian CSEC (Communications Security Establishment of Canada).

FIPS compliance support has been added to Cisco Unified Communications Manager. This mode can be enabled or disabled using CallManager Administration Command Line Interface.

From CiscoTSP, FIPS Compliance is supported by upgrading to FIPS 140-2 validated OpenSSL Version, “FIPS capable OpenSSL”, library which is used for setting TLS connection with CTI Manager or Cisco Unified Communications Manager. FIPS mode on CiscoTSP can be updated dynamically. Currently, CiscoTSP enables FIPS Mode and depends on FIPS Mode of the Cisco Unified Communications Manager.

Interface Changes

No interface changes.

Message Sequences

No impact on end user.

Backward Compatibility

This feature is backwards compatible.

Conference Changes

Forced Authorization Code and Client Matter Code

Cisco Unified TSP supports and interacts with two Cisco Unified Communications Manager features: Forced Authorization Code (FAC) and Client Matter Code (CMC). The FAC feature lets the System Administrator require users to enter an authorization code to reach certain dialed numbers. The CMC feature lets the System Administrator require users to enter a client matter code to reach certain dialed numbers.

The system alerts a user of a phone that a FAC or CMC must be entered by sending a “ZipZip” tone to the phone that the phone in turn plays to the user. Cisco Unified TSP will send a new LINE_DEVSPECIFIC event to the application whenever the application should play a “ZipZip” tone. Applications can use this event

to indicate when a FAC or CMC is required. For an application to start receiving the new LINE_DEVSPECIFIC event, it must perform the following steps:

1. lineOpen with dwExtVersion set to 0x00050000 or higher
2. lineDevSpecific – Set Status Messages to turn on the Call Tone Changed device specific events

The application can enter the FAC or CMC code with the lineDial() API. Applications can enter the code in its entirety or one digit at a time. An application may also enter the FAC and CMC code in the same string as long as they are separated by a "#" character and also ended with a "#" character. The optional "#" character at the end only serves to indicate dialing is complete.

If an application does a lineRedirect() or a lineBlindTransfer() to a destination that requires a FAC or CMC, Cisco Unified TSP returns an error. The error that Cisco Unified TSP returns indicates whether a FAC, a CMC, or both are required. Cisco Unified TSP supports two new lineDevSpecific() functions, one for Redirect and one for BlindTransfer, that allows an application to enter a FAC or CMC, or both, when a call gets redirected or blind transferred.

Forwarding

Cisco Unified TSP now provides added support for the lineForward() request to set and clear ForwardAll information on a line. This will allow TAPI applications to set the Call Forward All setting for a particular line device. Activating this feature will allow users to set the call forwarding Unconditionally to a forward destination.

Cisco Unified TSP sends LINE_ADDRESSSTATE messages when lineForward() requests successfully complete. These events also get sent when call forward indications are obtained from the CTI, indicating that a change in forward status has been received from a third party, such as Cisco Unified Communications Manager Administration or another application setting call forward all.

Gateway Recording

Cisco Unified Communications Manager has been providing a recording solution since release 6.0. In previous releases, the call recording was phone-based. The Cisco IP phones are used to fork, or direct, the two media streams of the agent-customer call to the recorder.

However, for call scenarios where the devices involved do not directly register with Unified Communications Manager, phone-based recording is not possible. This situation excludes the calls handled by the mobile agents from being recorded. In addition, the recording of mobile calls becomes increasingly mandatory by regulations in different jurisdictions, or becomes the essential business requirement for call centers or enterprises. These requirements call for a recording solution that does not rely on media forking from the endpoints.

The enhancements for the recording solution allow an external call that goes through a Cisco voice gateway to be recorded by having the voice gateway direct the two media streams to a voice recorder. This solution uses the existing CCiscoLineDevSpecificStartCallRecording and CCiscoLineDevSpecificStopCallRecording APIs to start and stop a user control recording session. Automatic recording is also configured on the line. There is an additional option to specify the voice gateway, or the IP phone as the prefered recording resource on a particular line.

In addition, the CTI Remote Device that was introduced in Cisco Unified Communications Manager Release 9.0 now supports recording through a gateway enabled for recording. If there is a recording gateway between the CTI Remote Device and the remote-destination where the call was routed, the recording can be started at

the CTI Remote Device. If there is a recording gateway between the caller and the CTI Remote Device, the recording can also be started at the CTI Remote Device. You can also configure the line(s) on the CTI Remote Device to support Automatic or Selective Recording. The CTI Remote Device can only use a gateway to route the media.

CTI port calls can be captured using the Network Recording feature available in Cisco Unified Communications Manager Release 10.0(1). The call media must pass through at least one recording-enabled gateway to be recorded. A typical use case: An external call to a CTI Port softphone. The Recording Media Source for the CTI Port is always gateway-preferred.

Interface Changes

- Recording Failure Event, on page 407
- LINECALLINFO, on page 320
- LINEDEVCAPS, on page 317

Message Sequences

Gateway Recording, on page 791

Backward Compatibility

Due to a design change to support Gateway Recording, the same recording may be stopped and restarted due to a feature invocation that was not seen in prior releases. Previously, this stop and start of the recording was only seen when the call being recorded is put on hold and then resume. In this release, this can happen when the party on the other end of the call being recorded changes. Take the following example:

Action	Pre-10.0	10.0
A calls B and B answers		
TSP application issues CCiscoLineDevSpecificStartCallRecording at A.	App receives SLDSMT_RECORDING_STARTED event at A	App receives SLDSMT_RECORDING_STARTED event at A
B transfers the call to C and C answers	No event is received at A	App receives SLDSMT_RECORDING_ENDED event at A App receives SLDSMT_RECORDING_STARTED event at A

Applications do not need to start the recording again but must handle these extra events.

Hold Reversion

The Hold Reversion feature allows a holding party to be notified about the HELD call after the hold reversion duration times out. The holding party gets audio and visible hold reversion notifications. The application can set the hold reversion event flag to receive the hold reversion notification from Cisco TSP. CallInfo and CallState of the call remains unchanged when a hold reversion event occurs and the LineCallDevSpecific event is sent to the application indicating the hold reversion if the application has enabled the hold reversion event flag.

Hunt List

CiscoTSP supports lines and their devices included in the Hunt List and provides appropriate information for applications to understand that the call is offered through a Hunt Pilot. Hunt List can include more than one Line Group and each Line Group may have different call distribution algorithms. Irrespective of the algorithm used in the Line Groups, CiscoTSP provides consistent information to applications.

When call offered on the line is routed through Hunt Pilot, CiscoTSP provides Hunt Pilot Directory Number or Partition in LINECALLINFO::DEVSPECIFIC for Caller, Called, or Connected IDs. However, there will be no Hunt Pilot information for Redirection and Redirecting ID. Hunt Pilot name is not passed to LINECALLINFO::DEVSPECIFIC.

There is no separate LineDevSpecific event to report the Hunt Pilot information, however, it is reported by existing event when caller, called, or connected id changes.

Hunt List can also interact with Call Pickup feature. However, Hunt List broadcast feature is not supported while interacting with Call Pickup feature. In this case, there will be no Pickup notification for the broadcasted call. CTI or Cisco Unified Communications Manager fails the pickup request if the application tries to pickup a broadcast call in Hunt List.

When call is routed to Hunt List and offering is accepted by Line Group member, the called information becomes Hunt Pilot's information. After Line Group member answers the call, connected ID will shows the actual Line Group member information with Hunt Pilot information in LINECALLINFO::DEVSPECIFIC. In this case, both called and connected ID contains Hunt Pilot information.

In case of a conference, Hunt Pilot Directory Number or Partition is presented if connected party's primary call is routed through Hunt List.

Note To support this feature, the application must negotiate line extension 0x00A0000 or above.

Interface Changes

LINECALLINFO, on page 320.

Message Sequences

See Hunt List, on page 802.

Backward Compatibility

This feature is backward compatible.

Hunt Pilot Connected Number

In Cisco Unified Communications Manager 9.0, the support for hunt pilots is enhanced to expose the huntmember which has answered the call as the called party in a call involving a hunt pilot at the calling side. With this enhancement, when a user calls a hunt pilot HP and the call is answered by the hunt list Line group member LG1, TAPI exposes DNof the HuntMember as the ModifiedConnectedParty DN under devspecific part of linecallinfo under Call Party Normalization info structure.

When this feature is disabled, the modifiedconnectedParty exposed is the HuntPilotDN.

The HuntPilot Information is available in the devspecific part of linecallinfo(under HuntPilotInfo structure). There is no change in Huntpilot information for call scenarios involving huntpilot, when this feature is enabled from the information exposed when this is feature is disabled.

Interface Changes

Not applicable.

Message Sequences

See Hunt Pilot Connected Number Feature, on page 866.

Backward Compatibility

This feature is backward compatible. It can be enabled on Huntpilot page **Display Line Group Member DN as Connected Party**. By default, this feature is disabled so that the existing applications are not affected.

Intercom

The Intercom feature allows one user to call another user and have the call automatically answered with one-way media from the caller to the called party, regardless of whether the called party is busy or idle.

To ensure that no accidental voice of the called party is sent back to the caller, Cisco Unified Communications Manager implements a 'whisper' intercom, which means that only one-way audio from the caller is connected, but not audio from the called party. The called party must manually press a key to talk back to the caller. A zip-zip (auto-answer) tone will play to the called party before the party can hear the voice of the caller. For intercom users to know whether the intercom is using one-way or two-way audio, the lamp for both intercom buttons appears colored amber for a one-way whisper Intercom and green for two-way audio. For TSP applications, only one RTP event occurs for the monitored party: either the intercom originator or the intercom destination, with the call state as whisper, in the case of a one-way whisper intercom.

TSP exposes the Intercom line indication and Intercom Speed Dial information in DevSpecific of LineDevCap. The application can retrieve the information by issuing LineGetDevCaps. In the DevSpecific portion, TSP provides information that indicates (DevSpecificFlag = LINEDEVCAPSDEVSPECIFIC_INTERCOMDN) whether this is the Intercom line. You can retrieve the Intercom speed dial information in the DevSpecific portion after the partition field.

If a CTI port is used for the Intercom, the application should open the CTI port with dynamic media termination. TSP returns LINEERR_OPERATIONUNAVAIL if the Intercom line is opened with static media termination.

Note You cannot use CTI Route Point for the Intercom feature.

The administrator can configure the speed dial and label options from Cisco Unified Communications Manager Administration. However, the application can issue CciscoLineSetIntercomSpeeddial with SLDST_LINE_SET_INTERCOM_SPEEDDIAL to set or reset SpeedDial and Label for the intercom line. The Application setting will overwrite the administrator setting that is configured in the database. Cisco Unified Communications Manager uses the application setting to make the intercom call until the line is closed or until the application resets it. In the case of a Communications Manager or CTIManager failover, CTIManager or Cisco TSP resets the speed dial setting of the previous application. If the application restarts, the application

must reset the speed dial setting; otherwise, Cisco Unified Communications Manager will use the database setting to make the intercom call. In any case, if resetting of the speed dial or label fails, the system sends a LINE_DEVSPECIFIC event to indicate the failure. When the application wants to release the application setting and have the speed dial setting revert to the database setting, the application can call CciscoLineSetIntercomSpeeddial with a NULL value for SpeedDial and Label.

If the speed dial setting is changed, whether due to a change in the database or because the application overwrites the setting, the system generates a LineDevSpecific event to indicate the change. However, the application needs to call CCiscoLineDevSpecificSetStatusMsgs with DEVSPECIFIC_SPEEDDIAL_CHANGED to receive this notification. After receiving the notification, the application can call LineGetDevCaps to find out the current settings of speed dial and label.

Users can initiate an intercom call by pressing the Intercom button at the originator or by issuing a LineMakeCall with a NULL destination if Speedial/Label is configured on the intercom line. Otherwise, LineMakeCall should have a valid Intercom DN.

For an intercom call, a CallAttribute field in LINECALLINFODEVSPECIFIC indicates whether the call is for the intercom originator or the intercom target.

After the intercom call is established, the system sends a zip-zip tone event to the application as a tone-changed event.

Users can invoke a TalkBack at the destination in two ways:

- By pressing the intercom button
- By issuing CciscoLineIntercomTalkback with SLDST_LINE _INTERCOM_TALKBACK

TSP reports the Whisper call state in the extended call state bit as CLDSMT_CALL_WHISPER_STATE. If the call is being put on hold because the destination is answering an intercom call by using talk back, the system reports the call reason CtiReasonTalkBack in the extended call reason field for the held call.

The application cannot set line features, such as set call forwarding and set message waiting, other than to initiate the intercom call, drop the intercom call, or talk back. After the intercom call is established, the system limits call features for the intercom call. For the originator, only LINECALLFEATURE_DROP is allowed. For the destination, the system supports only the LINECALLFEATURE_DROP and TalkBack features for the whisper intercom call. After the intercom call becomes two-audio after the destination initiates talk back, the system allows only LINECALLFEATURE_DROP at the destination.

Speed dial labels support unicode.

IPv6

The IPv6 support feature enables IPv6 capabilities in a Cisco Unified Communications Manager network. IPv6 increases the number of addresses available for network devices. TAPI can connect to Unified CM with IPv6 support if the IPv6 Support feature is enabled on Unified CM. IPv6 enhancements include the following:

- Provides the IPv6 address of the calling party to the called party in the Devspecific part of LINECALLINFO.
- Support to register a CTI port or a route point with an IPv6 address. The RTP destination address also contains IPv6 addresses if the same is involved in media establishment.

The TSP user interface includes the primary and backup CTI Manager address and a flag that indicates the preference of user while connecting to the CTI Manager. CTI ports and route points can be registered with IPv4, IPv6, or both.

The following new CiscoLineDevSpecific functions allow the application to specify IP address mode and IPv6 address before opening CTI port and route point:

- CciscoLineDevSpecificSetIPv6AddressAndMode
- CciscoLineDevSpecificSetRTPParamsForCallIPv6

For dynamic port registration, on receiving the SLDSMT_OPEN_LOGICAL_CHANNEL event, the CciscoLineDevSpecificSetRTPParamsForCallIPv6 allows the application to provide IPv6 information for the call.

Interface Changes

See Set IPv6 Address and Mode, on page 371.

Message Sequences

See IPv6 Use Cases, on page 891.

Backward Compatibility

This feature is backward compatible. The 0x00090000 extension must be negotiated to use this feature.

Transfer Changes

Join

In Cisco Unified Communications Manager, the Join softkey joins all the parties of established calls (at least two) into one conference call. The Join feature does not initiate a consultation call and does not put the active call on hold. It also can include more than two calls, which results in a call with more than three parties.

Cisco Unified TSP exposes the Join feature as a new device-specific function that is known as lineDevSpecific() – Join. Applications can apply this function to two or more calls that are already in the established state. This also means that the two calls do not need to be set up initially by using the lineSetupConference() or linePrepareAddToConference() functions.

Cisco Unified TSP also supports the lineCompleteTransfer() function with dwTransferMode = Conference. This function allows a TAPI application to join all the parties of two, and only two, established calls into one conference call.

Cisco Unified TSP also supports the lineAddToConference() function to join a call to an existing conference call that is in the ONHOLD state.

A feature interaction issue involves Join, Shared Lines, and the Maximum Number of Calls. The issue occurs when you have two shared lines and the maximum number of calls on one line is less than the maximum number of calls on the other line.

For example, in a scenario where one shared line, A, has a maximum number of calls set to 5 and another shared line, A', has a maximum number of calls set to 2, the scenario involves the following steps:

A calls B. B answers. A puts the call on hold.

C calls A. A answers. C puts the call on hold.

At this point, line A has two calls in the ONHOLD state, and line A' has two calls in the CONNECTED_INACTIVE state.

D calls A. A answers.

At this point, the system presents the call to A, but not to A'. This happens because the maximum calls for A' specifies 2.

A performs a Join operation either through the phone or by using the lineDevSpecific – Join API to join all the parties in the conference. It uses the call between A and D as the primary call of the Join operation.

Because the call between A and D was used as the primary call of the Join, the system does not present the ensuing conference call to A'. Both calls on A' will go to the IDLE state. As the end result, A' will not see the conference call that exists on A.

Join Across Lines (SCCP)

This feature allows two or more calls on different lines of the same device to be joined through the join operation. Applications can use the existing join API to perform the task. When the join across line happens, the consultation call on the different line on which the survival call does not reside will get cleared, and a CONFERENCED call that represents the consultation call will be created on the primary line where conference parent is created. This feature should have no impact when multiple calls are joined on the same line.

This feature is supported on SCCP devices that can be controlled by CTI. In addition, this feature also supports chaining of conference calls on different lines on the same device. Also, a join across line can be performed on a non-controller line; that is, the original conference controller was on a different device then where the join is being performed.

Note This feature returns an error if one of the lines that are involved in the Join Across Lines is an intercom line.

Backwards Compatibility

This feature is backward compatible.

Join Across Lines (SIP)

This feature allows two or more calls on different lines of the same device to be joined by using the join operation. Applications can use the existing join API to perform the task. When the join across line happens, the consultation call on the different line on which the survival call does not reside will get cleared, and a CONFERENCED call that represents the consultation call will get created on the primary line where conference parent is created. This feature should have no impact when multiple calls are joined on the same line.

This feature is supported both on SCCP and SIP devices that can be controlled by CTI. In addition, this feature also supports chaining of conference calls on different lines on the same device. Also, a join across line can be performed on a non-controller (the original conference controller was on a different device then where the join is being performed) line.

This feature returns an error if one of the lines involved in the Join Across Lines is an intercom line.

Interface Changes

None.

Message Sequences

See Join Across Lines, on page 897.

Backwards Compatibility

This feature is backward compatible.

Line-Side Phones That Run SIP

TSP supports controlling and monitoring of TNP-based phones that are running SIP. Existing phones (7960 and 7940) that are running SIP cannot be controlled or monitored by the TSP and should not get included in the control list. Though the general behavior of a phone that is running is similar to a phone that is running SCCP not all TSP features get supported for phones that are running SIP.

CCiscoPhoneDevSpecificDataPassThrough functionality does not support on phones that are running SIP configured with UDP transport due to UDP limitations. Phones that are running SIP must be configured to use TCP (default) if the CCiscoPhoneDevSpecificDataPassThrough functionality is needed.

TSP application registration state for TNP phones that are running SIP with UDP as transport may not remain consistent to the registration state of the phone. TNP phone that are running SIP with UDP as transport may appear to be registered when application reports the devices as out of service. This may happen when CTIManager determines that Unified CM is down and puts the device as out of service, but, because of the inherent delay in UDP to determine the lost connectivity, phone may appear to be in service.

The way applications open devices and lines on phones that are running SIP remains the same as that of phone that is running SCCP. It is the phone that controls when and how long to play reorder tone. When a SIP phone gets a request to play reorder tone, the phone that is running SIP releases the resources from Unified CM and plays reorder tone. The call appears to be IDLE to a TSP application even though reorder tone is being played on the phone. Applications can still receive and initiate calls from the phone even when reorder tone plays on the phone. Because resources have been released on Unified CM, this call does not count against the busy trigger and maximum number of call counters.

When consult call scenario is invoked on the SIP, the order of new call event (for consult call) and on hold call state change event (for original call).

Localization Infrastructure Changes

Beginning with Release 7.0(1), the TSP localization is automated. The TSP UI can download the new and updated locale files directly from a configured TFTP server location. As a result of the download functionality, Cisco TSP install supports only the English language during the installation.

During installation, the user inputs the TFTP server IP address. When the user opens the TSP interface for the **first time**, the TSP interface automatically downloads the locale files from the configured TFTP server and extracts those files to the resources directory. The languages tab in the TSP preferences UI also provides functionality to update the locale files.

Note To upgrade from Cisco Unified Communications Manager, Release 6.0(1) TSP to Cisco Unified Communications Manager, Release 7.0(1) TSP, you must ensure that Release 6.0(1) TSP was installed by using English. If Release 6.0(1) TSP is installed using any language other than English and if the user upgrades to Release 7.0(1) TSP, then the user must manually uninstall Release 6.0(1) TSP from Add/Remove programs in control panel and then perform a fresh install of Release 7.0(1) TSP.

Interface Changes

None.

Message Sequences

None.

Backward Compatibility

Only English locale is packaged in Cisco TSP installer. The TSP UI downloads the locale files from the configured TFTP server. The end user can select the required and supported locale after the installation.

Logical Partitioning

The Logical Partitioning feature restricts VoIP to PSTN calls and vice versa, based on the logical partitioning policy. Any request that interconnects a VOIP call to a PSTN call or vice versa in two different geographical locations fails and the error code is sent back to the applications.

The device, device pool, trunk, and gateway pages now provide configuration to select geo-location values and construction rules for geo-location strings.

A new enterprise parameter has been added for this feature with the following values:

- Name: Logical partitioning enabled
- Values: True or False
- Default: False

A new error code has been added for this feature: LINEERR_INVALID_CALL_PARTITIONING_POLICY 0xC000000C

Interface Changes

There are no interface changes for this feature.

Message Sequences

See Logical Partitioning, on page 912.

Backward Compatibility

This feature is backward compatible. To maintain earlier behavior, set the logical partitioning enabled parameter to **False**.

Message Waiting Indicator Enhancement

The Message Waiting Indicator (MWI) feature enhancement enables the application to display the following information on the supported phones:

- Total number of new voice messages (normal and high priority messages)
- Total number of old voice messages (normal and high priority messages)
- Number of new high priority voice messages
- Number of old high priority voice messages
- Total number of new fax messages (normal and high priority messages)
- Total number of old fax messages (normal and high priority messages)
- Number of new high priority fax messages
- Number of old high priority fax messages

MWI also includes two CCiscoLineDevSpecific subclasses are added to enhance the MWI functionality. Similar to the existing setMessageWaiting operation, one MWI operation sets the summary information for the controlled line, while the another MWI operation sets the message summary information on any line that is reachable by the controlled line, as defined by the configured calling search space of the controlled line.

Interface Changes

See Message Summary, on page 344 and Message Summary Dirn, on page 346.

Message Sequences

There are no message sequences for this feature.

Backward Compatibility

This feature is backward compatible.

Microsoft Windows Vista

Microsoft Windows Vista operating system supports Cisco TSP client with the following work around:

- Always perform the initial installation of the Cisco TSP and Cisco Unified Communications Manager TSP Wave Driver as a fresh install.
- If a secure connection to Cisco Unified Communications Manager is used, turn off/disable the Windows Firewall.
- If Cisco Unified Communications Manager TSP Wave Driver is used for inbound audio streaming, turn off/disable the Windows Firewall.

If Cisco Unified Communications Manager TSP Wave Driver is used for audio streaming, you must disable all other devices in the Sound, Video, and Game Controllers group.

Monitoring Call Park Directory Numbers

Cisco TSP supports monitoring calls on lines that represent Cisco Unified Communications Manager Administration Call Park Directory Numbers (Call Park DNs). Cisco TSP uses a device-specific extension in the LINEDEVCAPS structure that enables TAPI applications to differentiate Call Park DN lines from other lines. If an application opens a Call Park DN line, all calls that are parked to the Call Park DN are reported to the application. The application cannot perform any call-control functions on any of the calls at a Call Park DN.

In order to open Call Park DN lines, the Monitor Call Park DNs check box in the Cisco Unified Communications Manager Administration for the TSP user must be checked. Otherwise, the application will not see any of the Call Park DN lines upon initialization.

Multiple Calls Per Line Appearance

The following topics describe the conditions of Line Appearance.

Maximum Number of Calls

The maximum number of calls per Line Appearance remains database configurable, which means that the Cisco TSP supports more than two calls per line on Multiple Call Display (MCD) devices. An MCD device comprises a device that can display more than two call instances per DN at any given time. For non-MCD devices, the Cisco TSP supports a maximum of two calls per line. The absolute maximum number of calls per line appearance equals 200.

Busy Trigger

In Cisco Unified CM, a setting, busy trigger, indicates the limit on the number of calls per line appearance before the Cisco Unified CM will reject an incoming call. Be aware that the busy trigger setting is database configurable, per line appearance, per cluster. The busy trigger setting replaces the old call waiting flag per DN. You cannot modify the busy trigger setting using the CiscoTSP.

Call Forward No Answer Timer

Be aware that the Call Forward No Answer timer is database configurable, per DN, per cluster. You cannot configure this timer using the CiscoTSP.

New Cisco Media Driver

Cisco TSP now allows the application to use the new Cisco Media Driver (next generation Wave Driver). Cisco Media Driver provides applications with functions similar to the legacy kernel mode driver, has improved scalability, and supports latest Microsoft operating system releases.

Two additional device classes, ciscowave/in and ciscowave/out, have been introduced to support the new driver. These classes can be used in the lineGetID() to retrieve line-device identifiers for media devices associated with line when the new Cisco Media Driver is used for audio streaming. For more information, see Cisco TSP Media Driver, on page 411.

Interface Changes

None

Message Sequences

None

Backward Compatibility

This feature is not backward compatible.

Other-Device State Notification

TAPI specification does not have a provision to notify applications regarding state changes of non-opened devices. The Other-Device State Notification feature enhances Cisco TSP with that functionality. Using this feature, an application can assign devices that Cisco TSP should use to notify the application about non-opened device state changes.

Currently, Cisco TSP supports this feature only for line devices. An application can enable the feature on any open line device using a DEVSPECIFIC_OTHER_DEVICE_STATE_NOTIFY flag in CCiscoLineDevSpecificSetStatusMsgs lineDevSpecific request. Cisco TSP then uses this line to deliver other-device state change notifications to the application. Notifications are sent in LINE_LINEDEVSTATE message as follows:

```
LINE_LINEDEVSTATE
hDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) LINEDEVSTATE_OTHER;
dwParam2 = (DWORD) CISCO_LINEDEVSTATE_OTHER_REASON_constant;
dwParam3 = (DWORD) dwLineDeviceId;
```

CiscoLineDevStateOtherReason enumeration defines all relevant values of the Param2 in this notification. Param3 contains the TAPI identifier of the line device for which the state has changed.

The other-device state notification is a supplementary mechanism that can be used by other features that provide the application with state change notifications for non-opened devices. For example, EM Memory Optimization uses this feature to notify an application when a line device becomes active or inactive as a result of EM login or logout.

Interface Changes

CiscoLineDevStateOtherReason enumeration type provides details to LINEDEVSTATE_OTHER and is passed to the application as dwParam2 in the LINE_LINEDEVSTATE message

```
enum CiscoLineDevStateOtherReason
{
CiscoLineDevStateOtherReason_Unknown = 0,
CiscoLineDevStateOtherReason_OtherLineInactive,
CiscoLineDevStateOtherReason_OtherLineActive,
CiscoLineDevStateOtherReason_OtherLineCapsChange
};
```

DEVSPECIFIC_OTHER_DEVICE_STATE_NOTIFY flag in Set Status Messages, lineDevSpecific request must be used to enable other-device state notifications. For more information, see Set Status Messages, on page 349.

Message Sequences

See Extension Mobility Memory Optimization Option, on page 762.

Backward Compatibility

The other-device state notification is a supplementary feature intended to be used by other features that require to provide the application with state change notifications for non-opened devices. Its backward compatibility should be considered in a context of a specific feature. For more information, see Extension Mobility Memory Optimization Option, on page 63.

Park Monitoring

The Park Monitoring feature allows you to monitor the status of parked calls. This feature improves the user experience of retrieving the parked calls. When TAPI receives a parked call notification, a call representing the parked call is generated, and the call is set to CONNECTED INACTIVE state. The parked call is set to IDLE when it is retrieved or forwarded to Park Monitoring Forward No Retrieve Destination.

DEVSPECIFIC_PARK_STATUS event is sent when call is parked, reminded, retrieved, and aborted. LineDevSpecific SLDST_SET_STATUS_MESSAGES are enhanced to allow the application to enable/disable DEVSPECIFIC_PARK_STATUS event.

When Cisco TSP receives the LINE_PARK_STATUS event for the newly parked call, Cisco TSP simulates a call for each of the newly parked call using the SubID received from the LINE_PARK_STATUS event, and notifies the application about the new parked call using the LINE_NEWCALL event.

Cisco TSP uses LINE_CALLSTATE event to notify changes in the park status to the application. The park status in the LINE_CALLSTATE event can be one of the following:

- Parked -indicates a call is parked by the TSP monitored Cisco Unified IP phone.
- Retrieved -indicates a previously parked call is retrieved.
- Abandoned -indicates a previously parked call is disconnected while waiting to be retrieved.
- Reminder -indicates the park monitoring reversion timer for the parked call has expired.
- Forwarded -indicates the parked call has been forwarded to the configured Park Monitoring Forward No Retrieve destination, or if the FNR destination is not configured, the call is forwarded back to the parker.

When Cisco TSP receives the LINE_PARK_STATUS event, it maps the existing CALLINFO structure with the fields received from LINE_PARK_STATUS event. The application then retrieves the updated structure by invoking lineGetCallInfo.

The mapping of the fields in the LINE_PARK_STATUS event to the LINECALLINFO structure is as follows:

LINE_PARK_STATUS	LINECALLINFO--	Description
LineHandle	hline	Identifies the line handle to which this message applies

LINE_PARK_STATUS	LINECALLINFO--	Description
GCID	dwcallid	Identifies the global call handle to which this message applies.
TransactionID	dwRedirectingName	A unique ID that identifies a particular parked call
CallReason	dwReason	Identifies the call reason.
Park Status	dwBearerMode	Parked, retrieved, abandoned, reminder, forwarded -indicates the status of the parked call.
ParkSlotDN	dwCallerID	The park slot DN.
ParkSlotPartition	dwCallerIDName	The partition of the park slot DN.
ParkedPartyDN	dwCalledID	The parked party DN.
ParkedPartyPartition	dwCalledIDName	The partition of the parked party DN.

To maintain the existing behavior of the Park feature for the Cisco Unified IP Phones running SIP, you can set the value of the Park Monitoring Forward No Retrieve Destination timer equal to the existing Call Park Duration timer and leave the Park Monitoring Forward No Retrieve Destination blank.

To override the Park Monitoring feature for the Cisco Unified IP Phones running SIP, turn off the DEVSPECIFIC_PARK_STATUS message flag by using the lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

Interface Changes

See Set Status Messages, on page 349.

Message Sequences

See Park Monitoring, on page 925.

Backward Compatibility

This feature is backward compatible.

Partition

The CiscoTSP support of this feature will provide Partition support information. Prior to release 5.1, CiscoTSP only reported partial information about the DNs to the applications in that it would report the numbers assigned but not the information about the partitions with which they were associated.

Thus, if a phone has two lines that are configured with same DNs – one in Partition P1 and the other in P2, a TSP application would cannot distinguish between these two and consequently open only one line of these two.

CiscoTSP provides the partition information about all DNs to the applications. Thus, the preceding limitation gets overcome and applications can distinguish between and open two lines on a device, which share the same DN but belong to different partitions.

TSP applications can query for LINEDEVCAPS where the partition information is stored in the devSpecific portion of the structure. The application will receive the partition information for the CallerID, CalledID, ConnectedID, RedirectionID, and RedirectingID in a call. This is provided as a part of DevSpecific Portion of the LINECALLINFO structure.

Also, the partition information of the Call Park DN at which the call was parked is also sent to the applications. The value of the partition information is sent to applications in ASCII format.

Note

Password Expiry Notification

The password expiry notification notifies the user about the password expiry date and provides the specific reason for the initialization failure if the password is already expired or the account is locked. The user can create a credential policy and associate the same with user password credentials.

After the CiscoTSP application is started, the password expiry notification appears. This information appears only once, when the application is initially loaded, and will not be updated periodically.

If the application is already started and second application begins, the password expiry notification will not appear again because the application is already loaded.

CiscoTSP notifies the user about the password expiry as follows:

Popup message from CiscoTSP notifier: **Unified CM TSP initialization Success -Password will Expire in Days : 9**

Application Event Log message: **Information : Password will Expire in [9] Days**

Password Expired

The CiscoTSP initialization fails and a message is displayed if the password is expired.

The password of a user with Credential Policy configured can expire for any of the following reasons:

- User did not reset the password before the password expiry date.
- Credential Policy was not reconfigured to increase the number of days until password expiry.

Expired Password can be reset by either the administrator or the user.

CiscoTSP notifies the user about the expired password as follows:

Popup message from CiscoTSP notifier: **Unified CM TSP initialization failed -Password has Expired, Please RESET Password**

Application Event Log message: **ERROR : Provider Open Failed as Password is expired. User can RESET the Password**

Popup message from CiscoTSP notifier: **Unified CM TSP initialization failed -Password has Expired, User cannot RESET the Password, Please contact ADMIN**

Application Event Log message: **ERROR : Provider Open Failed as Password is expired. User cannot RESET the Password, Please contact ADMIN**

Account Lock

A user account gets locked in any of the following conditions:

- Threshold number of incorrect logins is exceeded. This appears as **Failed Logon** in the user credential page.
- Administrator has locked the user account.
- Credential has not been used in a number of days as specified onthe user credential page and the account is locked due to inactivity. This appears as **Inactive Days Allowed** on the user credential page.

CiscoTSP notifies the user about the initialization failure as follows:

Popup message from CiscoTSP notifier: **Unified CM TSP initialization failed -Account is LOCKED**.

Application Event Log message: **ERROR: Provider Open Failed as User Account is LOCKED**.

Interface Changes

No interface changes.

Message Sequences

There are no message sequences.

Backward Compatibility

Feature is backward compatible.

Privacy Release

The Cisco Unified Communications Manager Privacy Release feature allows the user to dynamically alter the privacy setting. The privacy setting affects all existing and future calls on the device.Cisco Unified TSP does not support the Privacy Release feature.

Redirect and Blind Transfer

The Cisco Unified TSP supports several different methods of Redirect and Blind Transfer. This section outlines the different methods as well as the differences between methods.

lineRedirect

This standard TAPI lineRedirect function redirects calls to a specified destination. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follows:

- Calling Search Space (CSS) — Uses CSS of the CallingParty (the party being redirected) for all cases unless the call is in a conference or a member of a two-party conference where it uses the CSS of the RedirectingParty (the party that is doing the redirect).
- Original Called Party — Not changed.

lineDevSpecific -redirect reset Original Called ID

This function redirects calls to a specified destination while resetting the Original Called Party to the party that is redirecting the call. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follow:

- Calling Search Space (CSS) — Uses CSS of the CallingParty (the party being redirected).
- Original Called Party — Set to the RedirectingParty (the party that is redirecting the call).

lineDevSpecific -redirect set Original Called ID

This function redirects calls to a specified destination while allowing the application to set the Original Called Party to any value. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follow:

- Calling Search Space (CSS) — Uses CSS of the CallingParty (the party being redirected).
- Original Called Party — Set to the preferredOriginalCalledID that the lineDevSpecific function specifies.

You can use this request to implement the Transfer to Voice Mail feature (TxToVM). Using this feature, applications can transfer the call on a line directly to the voice mailbox on another line. You can achieve TxToVM by specifying the following fields in the above request: voice mail pilot as the destination DN and the DN of the line to whose voice mail box the call is to be transferred as the preferredOriginalCalledID.

lineDevSpecific -redirect FAC CMC

This function redirects calls to a specified destination that requires either a FAC, CMC, or both. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follow:

- Calling Search Space (CSS) — Uses CSS of the CallingParty (the party being redirected).
- Original Called Party — Not changed.

lineBlindTransfer

Use the standard TAPI lineBlindTransfer function to blind transfer calls to a specified destination. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follow:

- Calling Search Space (CSS) — Uses CSS of the TransferringParty (the party that is transferring the call).
- Original Called Party — Set to the TransferringParty (the party that is transferring the call).

lineDevSpecific -blind transfer FAC CMC

This function blind transfers calls to a specified destination that requires a FAC, CMC, or both. The Calling Search Space and Original Called Party that Cisco Unified TSP uses for this function follow:

- Calling Search Space (CSS) — Uses CSS of the TransferringParty (the party that is transferring the call).

Original Called Party — Set to the TransferringParty (the party that is transferring the call).

Refer and Replaces for Phones That Are Running SIP

As part of CTI support for phones that are running SIP, TSP will support new SIP features Refer and Replaces. Refer, Refer with Replaces, Invite with Replaces represent different mechanisms to initiate different call control features. For example, Refer with Replaces in SIP terms can be translated to Transfer operation in Unified CM. Invite with Replaces can be used for Pickup call feature across SIP trunks. TSP will support event handling corresponding to the features that are initiated by Unified CM phones that are running SIP / third party phones that are running SIP. TSP will not support Refer / Replaces request initiation from the API. Because TAPI does not have Refer/Replaces feature related reason codes defined, the standard TAPI reason will be LINECALLREASON_UNKNOWN. TSP will provide new call reason to user as part of LINE_CALLINFO::dwDevSpecificData if the application negotiated extension version greater or equal to 0x00070000.

For In-dialog refer, TSP places Referrer in LINECALLSTATE_UNKNOWN | CLDSMT_CALL_WAITING_STATE call state with extended call reason as CtiCallReasonRefer. This helps present the Referrer's call leg such that applications cannot call any other call functions on this call. Any request on this call when it is in LINECALLSTATE_UNKNOWN | CLDSMT_CALL_WAITING_STATE will return an error as LINEERR_INVALCALLSTATE.

The Referrer must clear this call after the Refer request is initiated. If Referrer does not drop the call, Refer feature will clear this call if the refer is successful. LINECALLSTATE_IDLE with extended reason as CtiCallReasonRefer will get reported.

If Referrer does not drop the call and if Refer request fails (For example, Refer to target is busy), refer feature will restore the call between Referrer and Referee.

With Unified CM Phones that are running SIP, Unified CM makes all the existing call features transparent such that applications will get the existing events when the phone invokes a SIP feature whose behavior matches with the existing feature of Unified CM. For example, when Refer with Replaces is called by a phone that runs SIP (with both primary and secondary/consult call legs on same SIP line) within Unified CM cluster, all the events will get reported the same as Transfer feature.

Secure Conference

Prior to release 6.0(1), the security status of each call matched the status for the call between the phone and its immediately connected party, which is a conference bridge in the case of a conference call. No secured conference resource existed, so secure conference calls were not possible.

Release 7.0(1) supports a secured conference resource to enable secure conference. The Secure Conference feature lets the administrator configure the Conference bridge resources with either a non-secure mode or an encrypted signaling and media mode. If the bridge is configured in encrypted signaling and media mode, the Conference Bridge will register to Cisco Unified Communications Manager as a secure media resource. This

enables the user to create a secure conference session. When the media streams of all participants who are involved in the conference are encrypted, the conference exists in encrypted mode. Otherwise, the conference exists in non-secure mode.

The overall conference security status depends on the least-secure call in the conference. For example, suppose parties A (secure), B (secure), and C (non-secure) are in a conference. Even though the conference bridge can support encrypted sRTP and is using that protocol to communicate with A and B, C remains a non-secure phone. Thus, the overall conference security status is non-secure. Even though the overall conference security status is non-secure, because a secure conference bridge was allocated, all secure phones (in this case, A and B) will receive sRTP keys. TSP informs each participant about the overall call security status. The system provides the overall call security level of the conference to the application in the DEVSPECIFIC portion of LINECALLINFO to indicate whether the conference call is encrypted or non-secure.

The Secure Conference feature uses four fields to present the call security status:

```
DWORD CallSecurityStatusOffset;
DWORD CallSecurityStatusSize;
DWORD CallSecurityStatusElementCount;
DWORD CallSecurityStatusElementFixedSize;
```

The offset will point to following structure:

```
typedef struct CallSecurityStatusInfo
{
    DWORD CallSecurityStatus;
} CallSecurityStatusInfo;
```

The system updates LINECALLINFO whenever the overall call security status changes during the call because a secure or non-secure party joins or leaves the conference.

A conference resource gets allocated to the conference creator based on the creator security capability. If no conference resource with the same security capability is available, the system allocates a less-secure conference resource to preserve existing functionality.

When a shared line is involved in the secure conference, the phone that has its line in RIU (remote in use) mode will not show a security status for the call. However, TSP exposes the overall security status to the application along with other call information for the inactive call. This means that TSP also reports the OverallSecurityStatus to all RIU lines. The status will match what is reported to the active line. Applications can decide whether to expose the information to the end user.

Secure RTP

The secure RTP (SRTP) feature allows Cisco TSP to report SRTP information to application as well as allow application to specify SRTP algorithm IDs during device registration. The SRTP information that Cisco TSP provides will include master key, master salt, algorithmID, isMKIPresent, and keyDerivation. To receive those key materials, administrator needs to configure TLS Enabled and SRTP Enabled flag in Cisco Unified Communications Manager Admin User windows and establish TLS link between TSP and CTIManager.

Besides, during device registration, application can provide SRTP algorithm IDs for CTI port and CTI Route Point in case of media termination by application. Application should use new Cisco extension for Line_devSpecific -CciscoLineDevSpecificUserSetSRTPAlgorithmID to set supported SRTP algorithm IDs after calling LineOpen with 0x80070000 version or higher negotiated, then followed by either

CCiscoLineDevSpecificUserControlRTPStream or CciscoLineDevSpecificPortRegistrationPerCall to allow TSP to open device on CTI Manager.

When call arrives on an opened line, TSP will send LINE_CALLDEVSPECIFIC event to application with secure media indicator; then, application should query LINECALLINFO to get detail SRTP information if SRTP information is available. The SRTP information resides in the DevSpecific portion of the LINECALLINFO structure.

In case of mid-call monitoring, Cisco TSP will send LINE_CALLDEVSPECIFIC with secure media indicator, however there will be no SRTP information available for retrieval under this scenario. The event is only sent upon application request via PhoneDevSpecific with CPDST_REQUEST_RTP_SNAPSHOT_INFO message type.

To support SRTP that is using static registration, a generic mechanism for delayed device/line now exists. The following ones apply:

- Extension version bit SELSIUSTSP_LINE_EXT_VER_FOR_DELAYED_OPEN = 0x40000000
- CiscoLineDevSpecificType -SLDST_SEND_LINE_OPEN
- CCiscoLineDevSpecific -CciscoLineDevSpecificSendLineOpen

If application negotiates with 0x00070000 in lineOpen against CTI port, TSP will do LineOpen/DeviceOpen immediately. If application negotiates with 0x40070000 in LineOpen against CTI port, TSP will delay the LineOpen/DeviceOpen. Application can specify SRTP algorithm ID by using CciscoLineDevSpecificUserSetSRTPAlgorithmID (SLDST_USER_SET_SRTP_ALGORITHM_ID). However, to trigger actual device/line open in TSP, application needs to send CciscoLineDevSpecificSendLineOpen(SLDST_SEND_LINE_OPEN)

If application negotiates with 0x80070000 in LineOpen against CTI port/RP, TSP will delay the LineOpen/DeviceOpen until application specifies media information in CCiscoLineDevSpecific; however, application can use CciscoLineDevSpecificUserSetSRTPAlgorithmID (SLDST_USER_SET_SRTP_ALGORITHM_ID) to specify SRTP algorithm ID before specifying the media information.

If application negotiates with 0x40070000 in LineOpen against RP, TSP should fail CciscoLineDevSpecificUserSetSRTPAlgorithmID (SLDST_USER_SET_SRTP_ALGORITHM_ID) request because RP should have media terminated by application.

Currently, the SELSIUSTSP_LINE_EXT_VER_FOR_DELAYED_OPEN bit only gets used on CTI port when TSP Wave Driver is used to terminate media. Under conference scenario, the SRTP information gets stored in conference parent call for each party. An application that negotiates with correct version and is interested in the SRTP information in a conference scenario should retrieve the SRTP information from CONNECTED call of the particular conference party.

Backward Compatibility

CCiscoLineDevSpecific extension

CciscoLineDevSpecificUserSetSRTPAlgorithmID is defined.

CCiscoLineDevSpecific extension CciscoLineDevSpecificSendLineOpen is defined. An extra LINE_CALLDEVSPECIFIC event gets sent if negotiated version of application supports this feature while keeping existing LINE_CALLDEVSPECIFIC for reporting existing RTP parameters.

Wave driver (media terminating endpoint) uses the strip_policy to create a crypto context. It should match the encrypt and decrypt packets sent/received by IPPhones/CTIPorts. Phone uses one hardcoded srtp_policy for all phone types including phones that are using SIP.

policy->cipher_type = AES_128_ICM;

policy->cipher_key_len = 30;

policy->auth_type = HMAC_SHA1;

policy->auth_key_len = 20;

policy->auth_tag_len = 4;

policy->sec_serv = sec_serv_conf_and_auth;

Note Applications should not store key material and must use the proper security method to ensure confidentiality of the key material. Applications must clear the memory after key information is processed. Be aware that applications are subject to export control when they provide mechanism to encrypt the data (SRTP). Applications should get export clearance for that.

Presentation Indication

Secure TLS

Establishing secure connection to CTIManager involves application user to configure more information through Cisco TSP UI. This information will help TSP to create its own client certificate. This certificate is used to create a mutually authenticated secure channel between TSP and CTIManager.

TSP UI adds a new tab called Security and the options that are available on this tab follows:

- Check box for Secure Connection to CTIManager: If checked, TSP will connect over TLS CTIQBE port (2749); otherwise, TSP will connect over CTIQBE port (2748).
- Default setting specifies non secure connection and the setting will remain unchecked.

Ensure that the security flag for the TSP user is enabled through Cisco Unified Communications Manager Administration as well. CTIManager will perform a verification check whether a user who is connecting on TLS is allowed to have secure access. CTIManager will allow only security enabled users to connect to TLS port 2749 and only non secure users to connect to CTIQBE port 2748.

The user flag to enable security depends on the cluster security mode. If cluster security mode is set to secure, user security settings will have a meaning; otherwise, the connection has to be non secure. If secure connection to CTIManager is checked, the following settings will get enabled for editing.

- CAPF Server: CAPF server IP address from which to fetch the client certificate.
- CAPF Port: (Default 3804) – CAPF Server Port to connect to for Certificate download.
- Authorization Code (AuthCode): Required for Client authentication with CAPF Server and Private Key storage on client machine.

- Instance ID(IID): Each secure connection to CTIManager must have its own certificate for authentication. With the restriction of having a distinct certificate per connection, CAPF Server needs to verify that the user with appropriate AuthCode and IID is requesting the certificate. CAPF server will use AuthCode and IID to verify the user identity. After CAPF server provides a certificate, it clears the AuthCode to make sure only one instance of an app requests a certificate based on a single AuthCode. CCM admin will allow user configuration to provide multiple IID and AuthCode.
- TFTP Server: TFTP server IP address to fetch the CTL file. CTL, which file is required to verify the server certificate, gets sent while mutually authenticating the TLS connection.
- Check box to Fetch Certificate: This setting is not stored anywhere, instead only gets used to update the Client certificate when it is checked and will get cleared automatically.
- Number of Retries for Certificate Fetch: This indicates the number of retries TSP will perform to connect to CAPF Server for certificate download in case an error. (Default -0) (Range – 0 to 3)
- Retry Interval for Certificate Fetch: This will be used when the retry is configured. It indicates the (secs) for which TSP will wait during retries. (Default – 0) (Range – 0 to 15)

Because user is not expected to update the client certificate every time it changes, TSP UI will pop up a message when this box is checked by user that says “This will trigger a certificate update. Please make sure that you really want to update the TSP certificate now.” This will ensure that if user selects this check box in an error. TSP will fail to establish a secure connection to CTIManager if a valid certificate cannot be obtained. Each secure connection to CTIManager needs to have a unique certificate for authentication.

If an application tries to create more than one Provider simultaneously with the same certificate or when a session with the same certificate already exists/is open, CTI Manager disconnects both providers. TSP will try reconnecting to CTIManager to bring the provider in service. However, if both providers continuously try to connect by using the same duplicated certificate, both providers will be closed after a certain number of retries, and the certificate will be marked as compromised by CTIManager on Unified CM server. The number of retries after which the certificate should be marked as compromised is configurable from the CTIManager Service Parameter CTI InstanceId Retry Count. CTI manager rejects further attempt to open provider with the certificate that is compromised. In this case, the CAPF profile of the compromised certificate should be deleted and a new CAPF Profile must be created for the user. The new CAPF profile for the user should use new instance ID. Otherwise, the old certificate, which was compromised previously, can be used again.

A new error code, TSPERR_INIT_CERTIFICATE_COMPROMISED, with value as 0x00000011 where TSPERR_UNKNOWN is 0x00000010 now exists. Application should not have checks like “if (err < TSPERR_UNKNOWN))” because error codes exists that have a value greater than that.

When TLS is used, the initial handshake will be slow as expected due to heavy use of public key cryptography. After the initial handshake is complete and the session is established, the overhead gets significantly reduced. The following profiling result applies on ProviderOpen for both secure and non-secure CTI connection.

Controlled Devices	Type of CTI Connection	Duration on ProviderOpen	Duration on OpenAllLines	Comments
0	Non-Secure	1 sec 382 ms	N/A	
	Secure	4 sec 987 ms	N/A	With certificate retrieval.
	Secure	3 sec 736 ms	N/A	
100	Non-secure	1 sec 672 ms	3 sec 164ms	
	Secure	5 sec 758 ms	3 sec 445ms	

Controlled Devices	Type of CTI Connection	Duration on ProviderOpen	Duration on OpenAllLines	Comments
2500	Non-Secure	29 sec 513 ms	3 min 26 sec 728 ms	
	Secure	34 sec 219 ms	3 min 26 sec 928 ms	

Secured Monitoring and Recording

This feature enhances the ability to monitor or record calls in a secure environment. In Cisco Unified Communications Manager (Cisco Unified CM), Release 8.0(1), Cisco Unified CM software has been enhanced to enable call monitoring and recording in a secure environment. So, secured calls can be monitored and recorded.

The recording and monitoring session is created in a secure mode only when the Supervisor/Recording Device and the Agent has secure capabilities. Recording/Monitoring request is successful only when the Supervisor/Recording Device has higher than or meets the security capabilities of the Agent.

TransactionID, which is unique for reach monitoring session, is exposed to the application in Call Attribute information, DevSpecific part of Call Info for the call on supervisor and agent.

If the Silent Monitoring session is toned down when the Supervisor Security capabilities do not meet or exceed the capabilities of the agent, LINE_DEVSPECIFIC event is delivered with Param1 = SLDSMT_MONITORING_TERMINATED indicating the Monitoring Terminated event and Param2 = TransactionID of the call that is terminated.

To receive the Monitoring Terminated event, the DEVSPECIFIC_SILENT_MONITORING_TERMINATED message flag must be set in applications by using the lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

The application has to determine the monitoring session to be terminated based on the TransactionID that TSP exposes in LINE_DEVSPECIFIC Event for the Monitoring Session Terminated Event:

- Monitoring Terminated event is delivered to the original supervisor that initiated the Monitoring session and is longer present in monitoring the call.
- Recording: Cisco Unified CM does not support recording on Authenticated devices and also when the Recording device is authenticated.
- If the application monitors the Customer, Agent and Supervisor lines after Monitoring/Recording sessions start, CallReason will be LINECALLREASON_UNKNOWN for direct calls from the customer to the agent. CallReason will be LINECALLREASON_DIRECT for the monitored call on Supervisor as CTI reports the CallReason = CtiReasonSilentMonitoring/ CtiReasonRecording for respective Feature.

Note CallAttribute information in devspecific part of callInfo for a call on Supervisor is not cleared when the agent drops the call, in case the Monitored call is being conferenced by two Supervisors.

Interface Changes

New error Code – LINEERR_SECURITY_CAPABILITIES_MISMATCH 0xC000000E

Existing Cause Code – LINEDISCONNECTMODE_INCOMPATIBLE 0x00000400

New LINE_DEVSPECIFIC message Type -SLDSMT_MONITORING_TERMINATED. For more information, see Silent Monitoring Session Terminated Event, on page 407.

New LineDevSpecificSetStatusMessage Flag -DEVSPECIFIC_SILENT_MONITORING_TERMINATED. For more information, see Set Status Messages, on page 349.

In the CallAttributeInfo_ExtA0 field, LINECALLINFO::DEVSPECIFIC added the TransactionID field. For more information, see Details, on page 326.

Message Sequences

See Secure Monitoring and Recording, on page 976.

Select Calls

The Select softkey on IP phones lets a user select call instances to perform feature activation, such as transfer or conference, on those calls. The action of selecting a call on a line locks that call, so it cannot be selected by any of the shared line appearances. Pressing the “Select” key on a selected call will deselect the call.

Cisco Unified TSP does not support the “Select” function to select calls because all transfer and conference functions contain parameters that indicate on which calls the operation should be invoked.

Cisco Unified TSP supports the events that a user who selects a call on an application-monitored line causes. When a call on a line is selected, all other lines that share the same line appearance will see the call state change to dwCallState = CONNECTED and dwCallStateMode = INACTIVE.

Conference Changes

Transfer Changes

Set the Original Called Party Upon Redirect

Two extensions enable setting the original called party upon redirect as follows:

- CCiscoLineDevSpecificRedirectResetOrigCalled
- CCiscoLineDevSpecificRedirectSetOrigCalled

See lineDevSpecific, on page 148 for more information.

Shared Line Appearance

Cisco Unified TSP supports opening two different lines that each share the same DN. Each of these lines represents a Shared Line Appearance.

The Cisco Unified Communications Manager allows multiple active calls to exist concurrently on each of the different devices that share the same line appearance. The system still limits each device to, at most, one active call and multiple hold or incoming calls at any given time. Applications that use the Cisco Unified TSP to monitor and control shared line appearances can support this functionality.

If a call is active on a line that is a shared line appearance with another line, the call appears on the other line with the dwCallState = CONNECTED and the dwCallStateMode = INACTIVE. Even though the call party information may not display on the actual IP phone for the call at the other line, Cisco Unified TSP still reports the call party information on the call at the other line. This gives the application the ability to decide whether to block this information. Also, the system does not allow call control functions on a call that is in the CONNECTED INACTIVE call state.

Cisco Unified TSP does not support shared lines on CTI Route Point devices.

In the scenario where a line is calling a DN that contains multiple shared lines, the dwCalledIDName in the LINECALLINFO structure for the line with the outbound call may be empty or set randomly to the name of one of the shared DNs. The reason for this should be obvious as Cisco Unified TSP and the Cisco Unified Communications Manager cannot resolve which of the shared DN's you are calling until the call is answered.

Silent Install

The Cisco TSP installer supports silent install, silent upgrade, and silent reinstall from the command prompt. Users do not see any dialog boxes during the silent installation.

Silent Monitoring

Silent monitoring is a feature that enables a supervisor to listen to a conversation between an agent and a customer without the agent detecting the monitoring session. TSP provides monitoring type in line DevSpecific request for applications to monitor calls on a per call basis. Both monitored and monitoring party have to be in controlled list of the user.

The Application is required to send permanent lineID, monitoring Mode and toneDirection as input to CCiscoLineDevSpecificStartCallMonitoring. Only silent monitoring mode is supported and the supervisor cannot talk to the agent.

The Application can specify if a tone should be played when monitoring starts. ToneDirection can be none (no tone played), local (tone played to Agent only), remote (tone played to Customer and Supervisor), both local and remote (tone played to agent, customer, and supervisor).

```
enum PlayToneDirection
{
 PlayToneDirection_LocalOnly = 0,
 PlayToneDirection_RemoteOnly,
 PlayToneDirection_BothLocalAndRemote,
 PlayToneDirection_NoLocalOrRemote
};
```

Monitoring of a call which is in connected state on that line will start if the request is successful. This will result in a new call between supervisor and agent. However, the call will be automatically answered with Built-in Bridge (BIB) and agent is not be aware of the call. The call established between supervisor and agent will be one-way audio call. Supervisor will get the mixed stream of agent and customer voices. The application can only invoke the monitoring session for a call after it becomes active.

TSP will send LINE_CALLDEVSPECIFIC (SLDSMT_MONITORING_STARTED) event to the agent call when supervisor starts monitoring the call. TSP will provide monitored party's call attribute information (deviceName, DN, Partition) to the monitoring party in DEVSPECIFIC portion of LINECALLINFO after monitoring has started. Similarly, TSP will provide monitoring party's call attribute information (deviceName, DN, Partition) to the monitored party in devspecific data of LINECALLINFO after monitoring has started.

The monitoring session will be terminated when the agent-customer call is ended by either the agent or the customer. The supervisor can also terminate the monitoring session by dropping the monitoring call.TSP will inform agent by sending LINE_CALLDEVSPECIFIC (SLDSMT_MONITORING_ENDED) when supervisor drops the call. The event will not be sent if monitoring session has been ended after agent dropped the call.

SIP URL Address

As part of CTI support for phones that are using SIP, TSP will expose SIP URL that is received in Device/Call event that is received from CTIManager. The SIP URL will get presented for each corresponding party in extended call information structure of LINE_CALLINFO::dwDevSpecificData.

When a SIP trunk is involved in a call, the DN may not get presented in party information. Application then needs to consider SIP URL information under this call scenario for information.

TSP will provide SIP URL information to user as part of LINE_CALLINFO::dwDevSpecificData if the application negotiated extension version greater than or equal to 0x00070000.

CTI phones that are running SIP support the following features or functions:

- Call Initiate
- Call Answer
- Call Close/Disconnect
- Consult Transfer
- Direct Transfer
- Call Join
- Conference
- Hold/unhold
- Line Dial
- Redirect
- lineDevSpecific (SLDST_MSG_WAITING)
- lineDevSpecific (SLDST_MSG_WAITING_DIRN)

Presentation Indication

Change Notification of SuperProvider and CallPark DN Monitoring Flags

Super Provider

The Super Provider functionality allows a TSP application to control any CTI controllable device in the system (IP Phones, CTI Ports, CTI Route Points etc). The TSP application has to have an associated list of devices that it can control. It cannot control any devices that are outside of this list. However, certain applications would want to control a large number (possibly all) of the CTI controllable devices in the system. Super Provider enables the administrator to configure a CTI application as a "Super-Provider." This will mean that particular application can control absolutely any CTI controllable device in the system.

Previously, the Super Provider functionality was never exposed to TSP apps, that is the TSP application needed to have the device in the controllable list. In this release, TSP apps can control any CTI controllable device in the CallManager system. The Super-Provider apps need to explicitly "Acquire" the device before opening them.

TSP exposes new interfaces to the apps to explicitly Acquire/Deacquire any device in the system. The device information will be fetched for the explicitly acquired device using the SingleGetInfoFetch API exposed via QBE by CTI. Later, TSP will fetch the line information for this device using the LineInfoFetch API. The lines of this device are reported to the app using the LINE_CREATE/PHONE_CREATE events.

The apps can explicitly 'De-Acquire' the device. After the device is successfully de-acquired, TSP will fire LINE_REMOVE/PHONE_REMOVE events to the apps.

TSP also supports Change Notification of Super-Provider flag. If the administrator has configured a User as a Super-Provider in the admin pages, then the status of this is changed and the user is no more a Super-Provider, then CTI will inform the same to TSP in ProviderUserChangedEvent.

If any device had been explicitly acquired and opened in super-provider mode, then TSP will fire PHONE_REMOVE/LINE_REMOVE to the app indicating that the device/line is no more available for the app to use.

In this release, TSP supports change notification of CallParkDN Monitoring as well. Say, the user has been configured to allow monitoring of CallParkDN in the admin pages, now the status of this flag is disabled. Then TSP will fire LINE_REMOVE for the CallParkDNs.

Say, initially the CallParkDN Monitoring is disabled, now the status is changed to enabled, then TSP will fetch all the CallParkDNs from CTI and fire LINE_CREATE for each of the CallParkDNs.

SuperProvider

SuperProvider functionality allows a TSP application to control any CTI-controllable device in the system (IP Phones, CTI Ports, CTI Route Points and so on). Normally, a TSP application must have an associated

list of devices that it can control. It cannot control any devices that are outside this list; however, certain applications would want to control a large number (possibly all) the CTI controllable devices in the system.

SuperProvider functionality enables the administrator to configure a CTI application as a SuperProvider. This will mean that particular application can control absolutely any CTI controllable device in the system.

The SuperProvider functionality never gets exposed to TSP apps; that is, TSP application needed to have the device in the controllable list. In release 5.1 and later, TSP apps can control any CTI-controllable device in the Unified CM system.

The SuperProvider apps need to explicitly acquire the device before opening them. TSP exposes new interfaces to the apps to explicitly Acquire/Deacquire any device in the system. The device information is fetched for the explicitly acquired device by using the SingleGetInfoFetch API exposed via QBE by CTI. Later, TSP will fetch the line information for this device by using the LineInfoFetch API. The lines of this device are reported to the app by using the LINE_CREATE/PHONE_CREATE events.

The apps can explicitly 'De-Acquire' the device. After the device is successfully de-acquired, TSP will fire LINE_REMOVE/PHONE_REMOVE events to the apps.

TSP also supports Change Notification of "Super-Provider" flag. If the administrator has configured a User as a "Super-Provider" in the Unified CM Administration, the status of this changes and the user no longer represents a Super-Provider, then CTI will inform TSP in ProviderUserChangedEvent. If any device had been explicitly acquired and opened in super-provider mode, TSP will fire PHONE_REMOVE/LINE_REMOVE to the app and indicates that the device/line is no longer available for the app to use.

In release 5.1 and later, TSP supports change notification of CallParkDN Monitoring as well. If the user has been configured to allow monitoring of CallParkDN in the Unified CM Administration, the status of this flag is disabled. Then TSP will fire LINE_REMOVE for the CallParkDNs.

If the CallParkDN Monitoring is disabled, the status changes to enabled, TSP fetches all the CallParkDNs from CTI and fire LINE_CREATE for each of the CallParkDNs.

Support for Cisco Unified IP Phone 6900 and 9900 Series

In this release, CiscoTSP exposes Max Calls, Busy Trigger / Line Label, Line Instance, and Voice Mail Pilot Number in LineDevCap::DevSpecific interface.

TSP handles new device information -Device IP Address (IPv4 and IPv6) and NewCallRollOver/Consult call rollover/Join/DT/JAL/DTAL flag. This device information is kept in Device object and exposed through PhoneDevCap::DevSpecific, and also be exposed to Line App through LineDevCap::DevSpecific.

For NewCallRollOver/Consult call rollover/Join/DT/JAL/DTAL flag, there are two sets of information representing device setting and application behavior.

TAPI reports any change in the information above through LineDevSpecific event or PhoneDevSpecific event:

- Max Calls

  TAPI exposes Max Calls information in MaxCalls field of LineDevCaps::DevSpecific

  When the information changes, TSP reports the LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_MAX_CALLS bit on.

- Busy Trigger

  TAPI exposes busy trigger information in BusyTrigger field of LineDevCaps::DevSpecific

When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_BUSY_TRIGGER bit on.

- Line Instance ID

  TAPI exposes Line Instance ID (Line Button number) of the line configured on the device in LineInstanceNumber of LineDevCaps::DevSpecific

  When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_LINE_INSTANCE bit on.

- Line Label

  TAPI exposes Label of the line in LineLabelASCII and LineLabelUnicode field of LineDevCaps::DevSpecific

When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_LINE_LABEL bit on.

- Voice Mail Pilot

  TAPI exposes Voice Mail Box Pilot configured on the line in VoiceMailPilotDN field of LineDevCaps::DevSpecific

  When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_VOICEMAIL_PILOT bit on.

- Registered Device IP address and IP address mode

  TAPI exposes registered IP Address (IPv4 and IPv6) of the device in RegisteredIPv4Address and RegisteredIPv6Address fields of PhoneDevCaps::DevSpecific interface as well as in RegisteredIPv4Address and RegisteredIPv6Address fields of LineDevCaps::DevSpecific interface. Along with registered IP address, RegisteredIPAddressMode interface indicates whether the device is registered with IPv4, IPv6 or dual stack. If the device is unregistered, the RegisteredIPAddressMode has a value of IPAddress_Unknown. In case of IPAddress_Unknown, the RegisteredIPv4Address and RegisteredIPv6Address can be used only for reference.

  Device IP address applies only to IP phones and CTI Port and RP are not supported. When the information is changed, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_DEVICE_IPADDRESS bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_DEVICE_IPADDRESS bit on

- New Call Rollover

  TAPI exposes the new call rollover information configured on the device in NewCallRollOverEnabled flag of PhoneDevCaps::DevSpecific interface as well as in NewCallRollOverEnabled flag of LineDevCaps::DevSpecific interface. There are two sets of flags, one for device and one for application.

  When the information is changed, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_NEWCALL_ROLLOVER bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_NEWCALL_ROLLOVER bit on.

- Consult Call Rollover

TAPI exposes new call rollover information configured on the device in ConsultCallRollOverEnabled flag of PhoneDevCaps::DevSpecific interface as well as in ConsultCallRollOverEnabled flag of LineDevCaps::DevSpecific interface. There are two sets of flags, one for device and one for application.

When the information changes, TSP reports LineDevSpecific event with par am1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_CONSULTCALL_ROLLOVER bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_CONSULTCALL_ROLLOVER bit on.

- Join On Same Line

  TAPI exposes Join On Same Line information configured on the device in JoinOnSameLineEnabled flag of PhoneDevCaps::DevSpecific interface as well as in JoinOnSameLineEnabled flag of LineDevCaps::DevSpecific interface. There are two sets of flags, one for device and one for application.

  When changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_JOIN_ON_SAME_LINE bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_ PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_JOIN_ON_SAME_LINE bit on.

- Join Across Line

  TAPI exposes Join Across Line information configured on the device in JoinAcrossLineEnabled flag of PhoneDevCaps::DevSpecific interface as well as in JoinAcrossLineEnabled flag of LineDevCaps::DevSpecific interface. There are two set of flags, one for device and one for application.

  When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_JOIN_ACROSS_LINE bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_ PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_JOIN_ACROSS_LINE bit on.

- Direct Transfer On Same Line

  TAPI exposes Direct Transfer On Same Line information configured on the device in DirectTransferSameLineEnabled flag of PhoneDevCaps::DevSpecific interface as well as in DirectTransferSameLineEnabled flag of LineDevCaps::DevSpecific interface. There are two sets of flags, one for device and one for application.

  When the information changes, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_DIRECTTRANSFER_ON_SAME_LINE bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_DIRECTRANSFER_ON_SAME_LINE bit on.

- Direct Transfer Across Line

  TAPI exposes Direct Transfer Across Line information configured on the device in DirectTransferAcrossLineEnabled flag of PhoneDevCaps::DevSpecific interface as well as in DirectTransferAcrossLineEnabled flag of LineDevCaps::DevSpecific interface. There are two set of flags, one for device and one for application.

  When the information is changed, TSP reports LineDevSpecific event with param1 = SLDSMT_LINE_PROPERTY_CHANGED, param2 has LPCT_DIRECTTRANSFER_ACROSS_LINE bit on. For phone application, TSP reports PhoneDevSpecific event with param1 = CPDSMT_PHONE_PROPERTY_CHANGED_EVENT, param2 has PPCT_DIRECTRANSFER_ACROSS_LINE bit on. To receive the

PHONE_PROPERTY_CHANGED_EVENT, the application must use CCiscoPhoneDevSpecificSetStatusMsgs to set the DEVSPECIFIC_DEVICE_PROPERTY_CHANGED_EVENT bit.

For the change notification event described above, the event is delivered to the application by TAPI layer only if the line or device is opened (even though CisoTSP sends the event to TAPI layer). If the line or phone is not opened, the application should call LineGetDevCaps again to obtain latest information about the line/device. The new extension 0x00090001 must be opened or negotiated for this feature.

Interface Changes

See LINEDEVCAPS, on page 317, Line Property Changed Events, on page 404, and Set Status Msgs, on page 392.

Message Sequences

See Support for Cisco Unified IP Phone 6900 and 9900 Series Use Cases, on page 1016.

Backward Compatibility

This feature is backward compatible. To maintain backward compatibility new extension (0x00090001) is added.

Support for 100 + Directory Numbers

This feature enables users to have more than 100 Directory Numbers associated with a Device (Phones, CTI Ports and Route Points). TSP supports this feature and displays the corresponding Lines to the application.

Interface Changes

There are no interface changes.

Message Sequences

There are no message sequences.

Backward Compatibility

This feature is backward compatible.

Swap and Cancel Softkeys

The following softkeys have been added to the Cisco Unified IP Phone 7900 Series:

- Swap
- Cancel

Swap

The Swap softkey can be only be used when you use the Transfer or Conference feature. When you press Swap, the phone puts the consultative call on hold and resumes the primary call. Swap operation breaks the internal linkage between the primary and consultative calls, but you can still complete the transfer or conference operation.

Cancel

When you press Cancel before completing the transfer operation, the TSP receives an event notification from CTI and cancels any pending transfer or conference requests.

The Swap and Cancel features operate as follows:

- For swap operation, the primary call state is changed to CONNECTED, and the consult call state is changed to ONHOLD.
- For cancel operation, the primary call state is changed to ONHOLD, and consult call state remains as CONNECTED.
- To complete the transfer operation after swap or cancel, the application invokes LineCompleteTransfer or CciscoLineDevSpecificDirectTransfer.
- To complete the conference operation after swap or cancel, the application invokes Cisco Join API – CCiscoLineDevSpecificJoin.

When using the Swap and Cancel features, the Cisco Unified IP Phones maintain a consulting relationship between the primary and the consulting calls, on invoking consult transfer or consult conference:

- The Swap operation puts the active call on hold and retrieves the held call.
- The Cancel operation breaks the consulting relationship between the primary and the consulting calls.

When users perform the swap operation, the behavior remains the same while resuming calls and all pending transfer or conference operation are cancelled. Users can swap or toggle during consultative transfer or conference transactions, and also swap or toggle between call sessions during the transaction to check the status of each party.

Interface Changes

There are no interface changes for this feature.

Message Sequences

See Refer and Replace Scenarios, on page 960.

Backward Compatibility

This feature is backward compatible.

Translation Pattern

Warning TSP does not support the translation pattern because it may cause a dangling call in a conference scenario. The application needs to clear the call to remove this dangling call or simply close and reopen the line.

Presentation Indication

Change Notification of SuperProvider and CallPark DN Monitoring Flags

Unicode

Cisco TSP supports unicode character sets. TSP will send unicode party names to the application in all call events. The party name needs to be configured in Cisco Unified Communications Manager Administration. This support is limited to only party names. The locale information also gets sent to the application. The UCS-2 encoding for unicode gets used.

The party names will exist in the DevSpecific portion of the LINECALLINFO structure. In SIP call scenarios, where a call comes back into Unified CM from SIP proxy over a SIP trunk, only ASCII name will get passed because SIP has no way of populating both ASCII and unicode. As the result, the Connected and Redirection Unicode Name will get reported as empty to application.

Unrestricted Unified CM

Encryption is permanently disabled in Unrestricted Cisco Unified Communications Manager. In the current Cisco Unified Communications Manager, signaling and media encryption are configurable. Upgrading from the Unrestricted version to the Restricted version of Cisco Unified Communications Manager is not supported. Administrator cannot create a new role with security groups and roles (“Standard CTI Secure Connection” and “Standard CTI Allow Reception of SRTP Key Material”) as these roles are not available in Unrestricted Cisco Unified Communications Manager.

Note Upgrade from unrestricted version to the restricted version is not supported.

In case of an upgrade from Restricted to Unrestricted Cisco Unified Communications Manager, all the security features are disabled and Standard CTI Secure roles associated with the end user are removed. But the custom administrative user roles created with the CTI secure privileges mentioned above are not disabled in the Cisco Unified Communications Manager database. In such cases, the application can connect to Unrestricted Cisco

Unified Communications Manager as a non-secure application because CTIManager filters the information about CTI secure roles.

Also, after an upgrade from Restricted Cisco Unified Communications Manager to Unrestricted Cisco Unified Communications Manager, secure TAPI application cannot connect to Unrestricted Cisco Unified Communications Manager. To connect to the Unrestricted Cisco Unified Communications Manager after an upgrade, the application must disable secure connection from TSP UI.

If the application tries to register CTI Ports/Route Points as secure endpoints in an Unrestricted Cisco Unified Communications Manager, then the request fails. However, in some scenarios the registration request may pass, but the device remains closed and failure is reported to the application.

Interface Changes

There are no interface changes for this feature.

Message Sequences

See Unrestricted Unified CM, on page 1043.

Backward Compatibility

This feature is backward compatible.

URI Dialing

Cisco TSP supports dialing using directory URIs as the destination address. Cisco TSP uses the @ symbol to differentiate between directory URIs and directory numbers. If an @ symbol is present, the dialed address is a directory URI. Directory URIs can now be returned in the dwDevSpecificData call structure.

URI dialing is also supported for CTI Remote Devices. Remote destinations can be configured with directory URIs as the remote destination number.

Interface Changes

The following interfaces have changed to support directory URIs:

- lineMakeCall—lpszDestAddress parameter can contain a directory URI
- lineBlindTransfer—lpszDestAddress parameter can contain a directory URI
- lineForward—dwDestAddressOffset in the lineForward structure can now point at a destination address
- lineRedirect—lpszDestAddress parameter can contain a directory URI

Message Sequence Charts

There is no change to the message sequence.

Backwards Compatibility

No backwards compatibility issues.

Video On Hold Support

CiscoTSP is enhanced to provide capability for applications to select/Play Video file when call is placed on Hold. This enhancement was designed for the Remote Expert solution. The newly added contentID is a pass through from application (TAPI) to CCM. TAPI will not process or manipulate this value. The contentID references a VoH stream to be played when the call is placed on hold.

The VoH files are housed externally on a media sense server. To have video on hold capability, the video on hold server must be configured in CCM Admin. This server coincides to the media sense server which houses all the VoH files.

New CCiscoLineDevSpecific extension - "**CciscoLineDevSpecificHoldEx**" is added to support this capability for the applications.

Interface Changes

See lineHold Enhancement, on page 385.

Use Cases

See LineHold Enhancement, on page 1045.

Backward Compatibility

There are no backward compatibility issues for this feature.

Whisper Coaching

Silent Call Monitoring is a feature that allows a supervisor to discreetly listen to a conversation between an agent and a customer without allowing the agent to detect the monitoring session. Whisper Coaching is an enhancement to the Silent Call Monitoring feature. Whisper Coaching allows supervisors to talk to agents during a monitoring session. This feature provides applications the ability to change the current monitoring mode of a monitoring call from Silent Monitoring to Whisper Coaching and vice versa.

If the application opens the line while the whisper coaching session is already in progress, TSP exposes either the TSPCallAttribute_WhisperCoachingCall or TSPCallAttribute_WhisperCoachingCall_Target bitmap in the CallAttributeType field of LineCallInfo::DevSpecific. This indicates whether the call is a whisper coaching call or the target of a whisper coaching call.

Support of this feature consists of the following:

- In the exiting CciscoLineDevSpecificStartCallMonitoringrequest, an m_MonitorMode parameter is enhanced to support an enumeration for Whisper coaching.
- A new CCiscoLineDevSpecific extension, CciscoLineDevSpecificMonitoringUpdateMode, needs to be added that allows the application to toggle between the silent monitoring and whisper coaching modes and vice versa.
- A new LineCallDevSpecific event updates the application stating that the monitoring mode has changed.
- A new field, activeToneDirection, is added in the CallAttributeInfo_ExtA0 structure. This field specifies the active tone direction that is being played for the call.

The behaviors of toneDirection are as follows:

- The existing service parameters for toneDirection are used for both silent and whisper coaching monitoring sessions. The service parameters are “Play Monitoring Notification Tone To Observed Target” (for a local party or an agent) and “Play Monitoring Notification Tone To Observed Connected Parties” (for a remote party or a customer).
- For CciscoLineDevSpecificStartCallMonitoring (monitoring mode = Silent) or CciscoLineDevSpecificMonitoringUpdateMode (monitoring mode = Silent), the application specified toneDirection is used in addition to the toneDirection configured with the service parameters.
- For CciscoLineDevSpecificStartCallMonitoring (monitoring mode = Whisper) or CciscoLineDevSpecificMonitoringUpdateMode (monitoring mode = Whisper), the application specified toneDirection for a remote side (customer only) is honored. This feature uses the service parameter that configured play toneDirection and the application specified toneDirection to play the tone to the remote side (customer only).

The following table lists effective toneDirection for StartSilentMonitoring/Toggle to SilentMonitoring.

Table 1: Effective ToneDirection for StartSilentMonitoring/Toggle to SilentMonitoring

	Observed Target = false **Observed Connected Parties = false**	**Observed Target = true** **Observed Connected Parties = false**	**Observed Target = false** **Observed Connected Parties = true**	**Observed Target = true** **Observed Connected Parties = true**
AppRequests toneDirection = None	None	Local Only	Remote Only	Both
AppRequests toneDirection = Local Only	Local Only	Local Only	Both	Both
AppRequests toneDirection = Remote Only	Remote Only	Both	Remote Only	Both
AppRequests toneDirection = Both	Both	Both	Both	Both

The following table lists effective toneDirection for Start WhisperCoaching/Toggle to WhisperCoaching.

Table 2: Effective ToneDirection for StartWhisperCoaching/Toggle to WhisperCoaching

	Observed Target = false **Observed Connected Parties = false**	**Observed Target = true** **Observed Connected Parties = false**	**Observed Target = false** **Observed Connected Parties = true**	**Observed Target = true** **Observed Connected Parties = true**
AppRequests toneDirection = None	None	None	Remote Only	Remote Only
AppRequests toneDirection = Local Only	None	None	Remote Only	Remote Only

	Observed Target = false Observed Connected Parties = false	Observed Target = true Observed Connected Parties = false	Observed Target = false Observed Connected Parties = true	Observed Target = true Observed Connected Parties = true
AppRequests toneDirection = Remote Only	Remote Only	Remote Only	Remote Only	Remote Only
AppRequests toneDirection = Both	Remote Only	Remote Only	Remote Only	Remote Only

Due to the fix for CSCsb89374, the behavior for the toggle request using CciscoLineDevSpecificMonitoringUpdateMode is different between SIP and SCCP phones. When CciscoLineDevSpecificMonitoringUpdateMode is changed:

- For SCCP phones, no media break is reported
- For SIP phones, a media break is always reported

Consider the following example for toggle:

```
dwParam1 = 0x30403 represents StartReception
dwParam1 = 0x e0500401 represents StartTransmission
dwParam1 = 0x4 represents StopReception
dwParam1 = 0x2 represents StopTransmission
```

	SIP	SCCP
Toggle from Silent to Whisper	LINE_DEVSPECIFIC dwParam1 = 0x4 LINE_DEVSPECIFIC dwParam1 = 0x30403 LINE_DEVSPECIFIC dwParam1 = 0x e0500401	LINE_DEVSPECIFIC dwParam1 = 0xe0500401
Toggle from Whisper to Silent	LINE_DEVSPECIFIC dwParam1 = 0x4 LINE_DEVSPECIFIC dwParam1 = 0x2 LINE_DEVSPECIFIC dwParam1 = 0x30403	LINE_DEVSPECIFIC dwParam1 = 0x2

Interface Changes

See UpdateMonitorMode, on page 381, Monitor Mode Update Event, on page 409, Details, on page 326, Details, on page 326, and Details, on page 326.

Message Sequences

See Whisper Coaching, on page 1045.

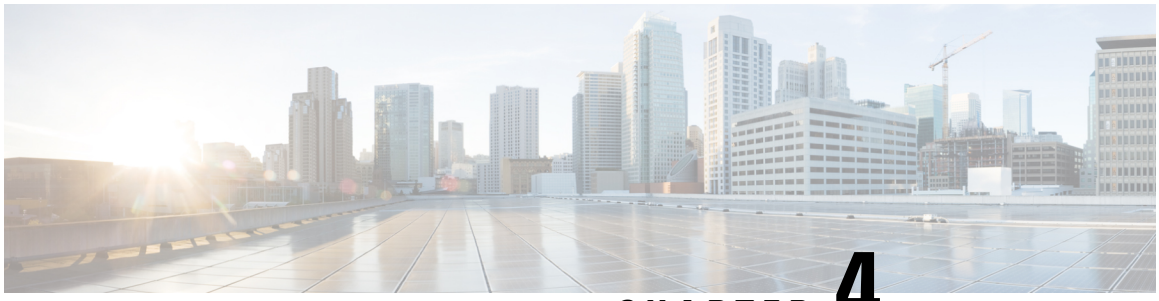
Backward Compatibility

This feature is backward compatible.

XSI Object Pass Through

XSI-enabled IP phones allow applications to directly communicate with the phone and access XSI features, such as manipulate display, get user input, play tone, and so on. To allow TAPI applications access to the XSI capabilities without having to set up and maintain an independent connection directly to the phone, TAPI provides the ability to send the device data through the CTI interface. The system exposes this feature as a Cisco Unified TSP device-specific extension.

The system only supports the PhoneDevSpecificDataPassthrough request for IP phone devices.



CHAPTER 4

Cisco Unified TAPI Installation

This chapter describes how to install and configure the Cisco Unified Telephony Application Programming Interface (TAPI) client software for Cisco Unified Communications Manager.

Note The upgraded TAPI client software does not work with previous releases of Cisco Unified Communications Manager.

Required Software

Cisco TSP requires the following software:

- Cisco Unified Communications Manager
- Supported Windows Platforms, on page 105

Supported Windows Platforms

All Windows operating systems support Cisco TAPI. Depending on the type and version of your operating system, you may need to install a service pack.

For a detailed breakdown of supported Windows platforms for Cisco Unified TAPI, see https://developer.cisco.com/site/tapi/documents/supported-windows-os/ .

Caution You can install Cisco TSP on a system only where Windows is installed on C:\. If Windows is installed on a drive other than C:\, an attempt to install Cisco TSP will fail.

Note Windows(64-bit) Operating Systems require native 64-bit Cisco TSP client. For more information on availability, see http://developer.cisco.com/web/tapi/blogroll.

Note Cisco TSP legacy wave driver is not supported under VMWare.

.

Installing the Cisco Unified CM TSP Client

Download the Cisco TSP client software from the Cisco Unified CM Administration Plug-Ins page. For information on installing plug-ins, refer to the Cisco Unified Communications Manager Administration Guide. The Cisco TSP client installation wizard varies depending on whether previous versions have been installed.

Note If you are installing multiple TSPs, multiple copies of CiscoTSPXXX.tsp and CiscoTUISPXXX.dll files will exist in the same Windows system directory.

To begin installation of the Cisco TSP client from the Cisco Unified CM Administration, perform the following steps:

Procedure

1. Download the Cisco Telephony Service Provider plugin from Cisco Unified Communication Manager **Administration** > **Application** > **plugins**.
2. Save it on the desired Desktop.
3. Double-click Cisco TSP.exe.
4. Follow the online instructions.

Cisco TSP Client Interaction with Windows Services

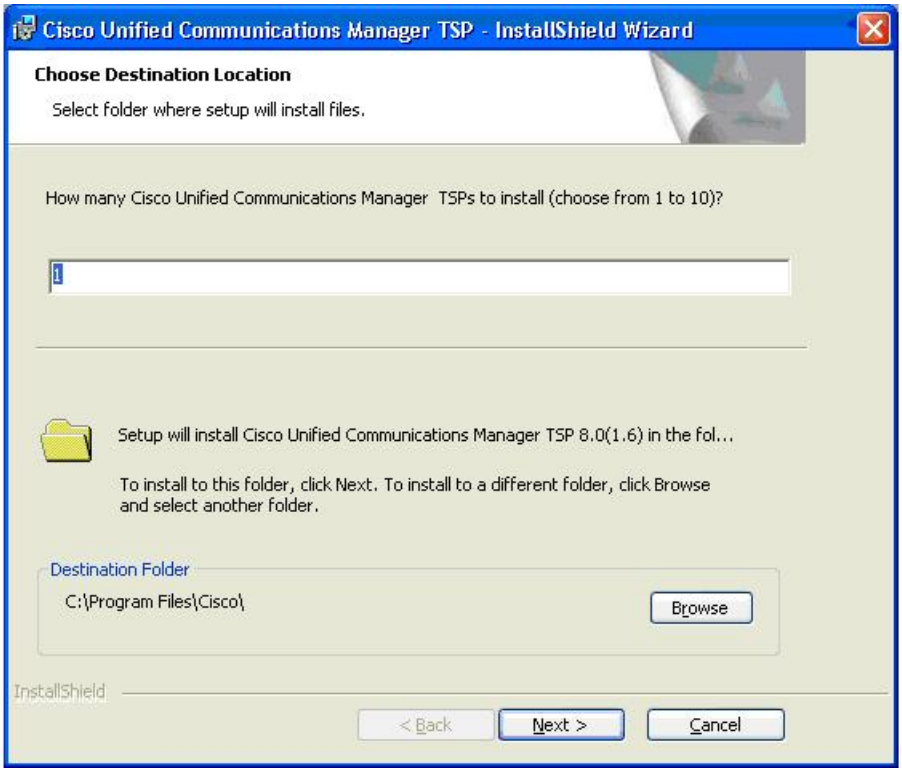
The Cisco TSP client is tightly coupled with the Microsoft Telephony and QWAVE Services. These services should not be stopped as it will lead to unexpected and undesirable behaviour of CiscoTSP . By default the Microsoft Telephony Service is dependent on the Microsoft Remote Connection Manager Service which cannot be restarted manually. To ensure proper installation of the Cisco TSP client, reboot the computer following Cisco TSP installation. Cisco also recommends rebooting the computer anytime the Cisco TSP configuration settings are changed to ensure the Microsoft Telephony Service is updated properly. If the

Microsoft Remote Connection Service is disabled, computer reboot is not required; simply restart the Microsoft Telephony Service using the Windows Services Applet for the new settings to take effect.

Installation Setup Screen

Click on Cisco TSP.exe to begin the installation process. Specify the destination folder where the Cisco TSP files must reside from the Choose Destination Location screen (shown below) and configure the number of TSP instances desired.

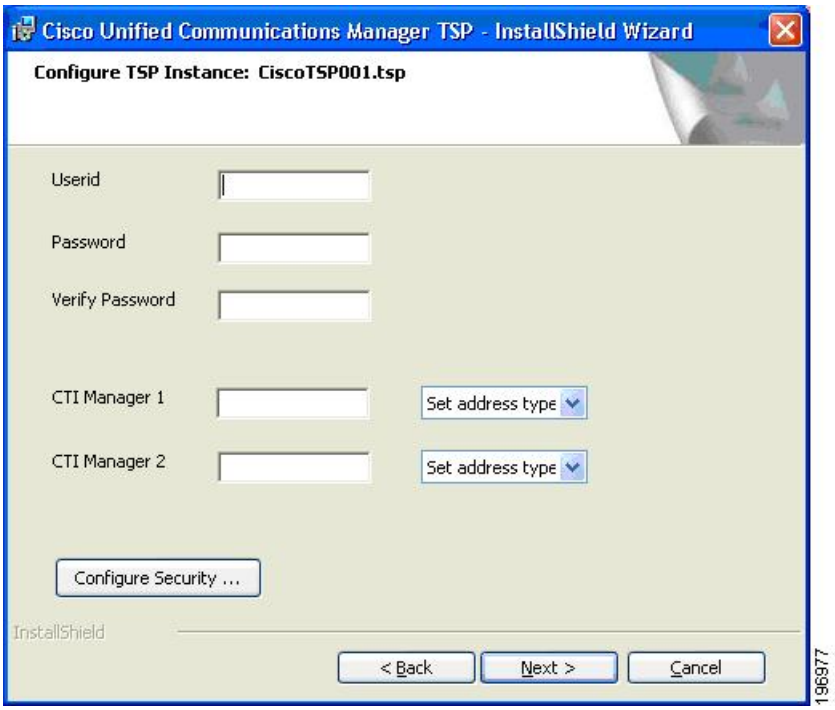
Figure 11: Choose Destination Location Screen



Configure TSP Instance

For each desired TSP Instance, configure the User ID, Password, and CTIManager settings in the Configure TSP Instance screen shown below. If the required information is not known during the installation, it can be configured later using the **Open Cisco TAPI Configuration** icon found in the Programs Menu (available after the installation).

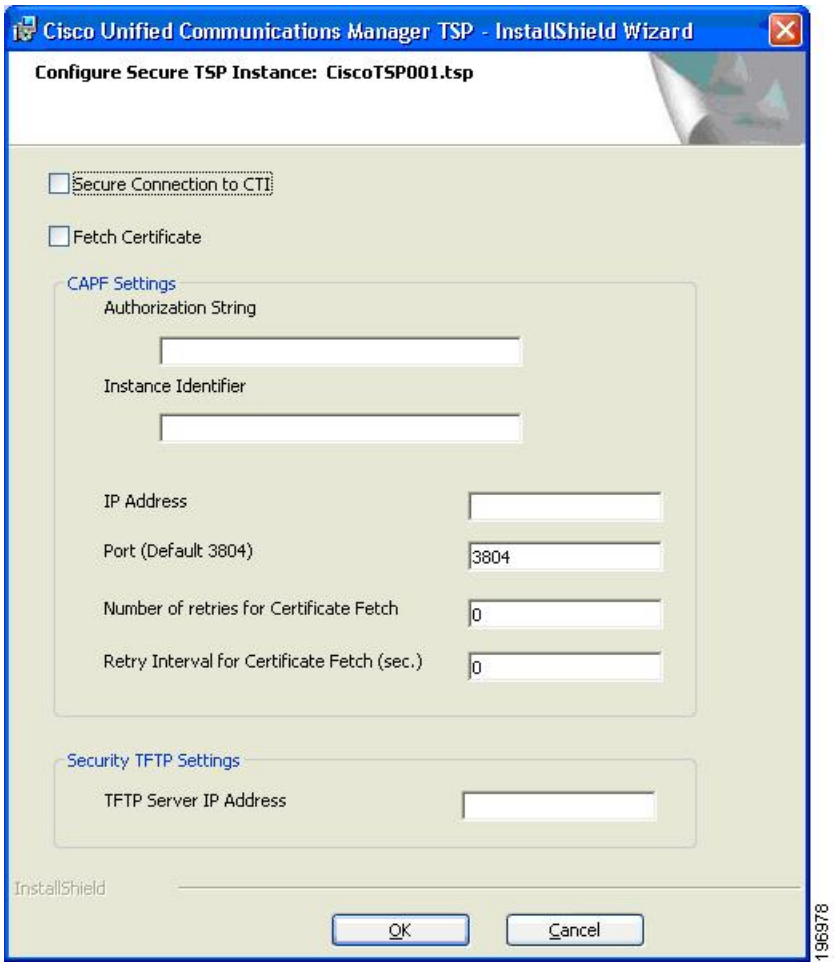
Figure 12: Configure TSP Instance Screen



Configure Secure TSP Instance

To configure a secure CTI connection, click the **Configure Security** button on the desired Configure TSP Instance. Populate the specified information from the Configure Secure TSP Instance screen as shown below. Refer to the Cisco Unified Communications Manager Security Guide for the desired release for additional information regarding Securing CTI.

Figure 13: Configure Secure TSP Instance Screen



Cisco Media Driver Selection

Cisco introduced a new media driver in Cisco Unified Communications Manager 8.0(1). The TSP client may be configured to install the Cisco Media Driver or the Cisco Wave Driver. Cisco Media and Wave Drivers are used with TSP applications that play or record media. Refer to the Installation Guide provided by your vendor to determine which driver may be required.

Note Cisco Media Driver settings apply to all configured TSP instances. Cisco Wave Driver settings are instance specific.

Cisco Media Driver

To install the Cisco Media Driver, select Cisco Media Driver from the installation screen. Set the desired start and end ports used by Cisco Media Driver. The port settings are used by all TSP instances. Each media channel requires 4 ports (1 Channel = 4 ports). Refer to the Installation Guide provided by your application vendor to

determine the appropriate port settings. The Media Driver is ready for use after the installation completes and the computer is rebooted.

Cisco Wave Driver

Select Cisco Wave Driver from the installation screen. Complete the TSP installation and reboot the PC. After the PC has rebooted, complete the installation of the Cisco Wave Driver by performing the following steps.

Cisco Wave Driver for Windows XP, Vista, 2003, 2008

Procedure for Windows XP / Windows Vista / Windows 2003 / Windows 2008

Procedure

Step 1 Open the Control Panel.

Step 2 Open **Add Hardware**. The Add Hardware Wizard window appears.

Step 3 Click **Next**.

Step 4 Select **Yes, I have already connected the hardware**.

Step 5 Select **Add a New Hardware Device**.

Step 6 Click **Next**.

Step 7 Select **Install the Hardware that I manually select from a list**.

Step 8 Click **Next**.

Step 9 For the hardware type, choose **Sound, video and game controller**.

Step 10 Click **Next**.

Step 11 Click **Have Disk**.

Step 12 Click **Browse** and navigate to the Wave Drivers folder in the folder where the Cisco Unified Communication Manager TSP is installed.

Step 13 Choose **OEMSETUP.INF** and click **Open**.

Step 14 In the Install From Disk window, click **OK**.

Step 15 In the Select a Device Driver window, select the **Cisco Unified Communication Manager TAPI Wave Driver** and click **Next**.

Step 16 In the Start Hardware Installation window, click **Next**.

Step 17 If Prompted for Digital signature Not Found, click **Continue Anyway**.

Step 18 The installation may issue the following prompt:

The avaudio32.dll file on Windows NT Setup Disk #1 is needed,

Type the path where the file is located and then click **OK**.

If so, navigate to the same location where you chose OEMSETUP.INF, select avaudio32.dll, and click **OK**.

Step 19 Click **Yes**.

Step 20 Click **Finish**.

Step 21 Click **Yes** to restart the computer.

Cisco Wave Driver for Windows 7

Procedure for Windows 7

Procedure

Step 1 Right click **My computer** and select **Manage**. The Computer Management window appears.

Step 2 From **System Tools**, select **Device Manager**.

Step 3 Right click on the <Computer-Name> and select **Install Legacy Wave Driver**. This action pops up an Add Hardware window.

Step 4 Select **Install the Hardware that I manually select from a list**.

Step 5 Click **Next**.

Step 6 For the hardware type, choose **Sound, video and game controller**.

Step 7 Click **Next**.

Step 8 Click **Have Disk**.

Step 9 Click **Browse** and navigate to the Wave Drivers folder in the folder where the Cisco Unified Communication Manager TSP is installed.

Step 10 Choose **OEMSETUP.INF** and click **Open**.

Step 11 In the Install From Disk window, click **OK**.

Step 12 In the Select a Device Driver window, select the **Cisco Unified Communication Manager TAPI Wave Driver** and click **Next**.

Step 13 In the Start Hardware Installation window, click **Next**.

Step 14 If Prompted for Digital signature Not Found, click **Continue Anyway**.

Step 15 The installation may issue the following prompt:

The avaudio32.dll file on Windows NT Setup Disk #1 is needed,

Type the path where the file is located and then click **OK**.

If so, navigate to the same location where you chose OEMSETUP.INF, select avaudio32.dll, and click **OK**.

Step 16 Click **Yes**.

Step 17 Click **Finish**.

Step 18 Click **Yes** to restart the computer.

Verifying the Cisco Wave Driver

Use these steps to verify the Cisco Wave Driver when performing install and uninstall operations.

Procedure

Step 1 Click **Start** > **Run**.

Step 2 In the text box, enter regedit.

Step 3 Click **OK**.

Step 4 Choose the Drivers32 key that is located in the following path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion
```

Step 5 If you are installing the Wave driver, make sure that the driver avaudio32.dll displays in the data column. If you are uninstalling the Wave driver, make sure that the driver avaudio32.dll does not display in the data column. This designates the Cisco Wave Driver.

Step 6 Verify that the previously existing Wave values appear in the data column for Wave1, Wave2, Wave3, and so on. You can compare this registry list to the contents of the .reg file that you saved in the procedure by opening the .reg file in a text editor and viewing it and the registry window side by side.

Step 7 If necessary, add the appropriate WaveX string values for any missing Wave values that should be installed on the system. For each missing Wave value, choose **Edit** > **New** > **String Value** and enter a value name. Then, choose **Edit** > **Modify**, enter the value data, and click **OK**.

Step 8 Close the registry by choosing **Registry** > **Exit**.

Configure the Cisco Wave Driver settings using the Wave tab in the Cisco TAPI Configuration tool (available after installation).

AutoUpgrade

The TSP client can be configured to detect and install new client versions automatically when Cisco Unified CM is upgraded. When set to Always or Ask, the auto-upgrade service requires the login User to have local Administrative rights to install applications. If the logged-in User is not permitted to install applications, set Auto-upgrade to Never.

Update Credentials

Cisco TSP 8.0 introduces a new feature that enables the Administrator to allow Users to update their credentials (UserID and Password) without requiring the User to have local Administrative rights. When this option is checked, all Standard Users can update their UserID and Password using the Cisco TAPI Configuration tool. Configuring all other Cisco TSP options requires local Administrative rights. If the Administrator does not want to allow Standard Users to update their credentials, leave this unchecked.

Cisco TSP Notifier

Cisco TSP 8.0 introduces a new feature that helps identify connectivity issues between Cisco Unified CM CTI Manager and the PC where the TSP client has been installed. TSP Notifier is installed automatically and can be configured to run during Windows start-up. The user must disable UAC to have the notifier run during startup. TSP Notifier runs as a background application in the system tray.

The Cisco TAPI Notifier can detect the following errors:

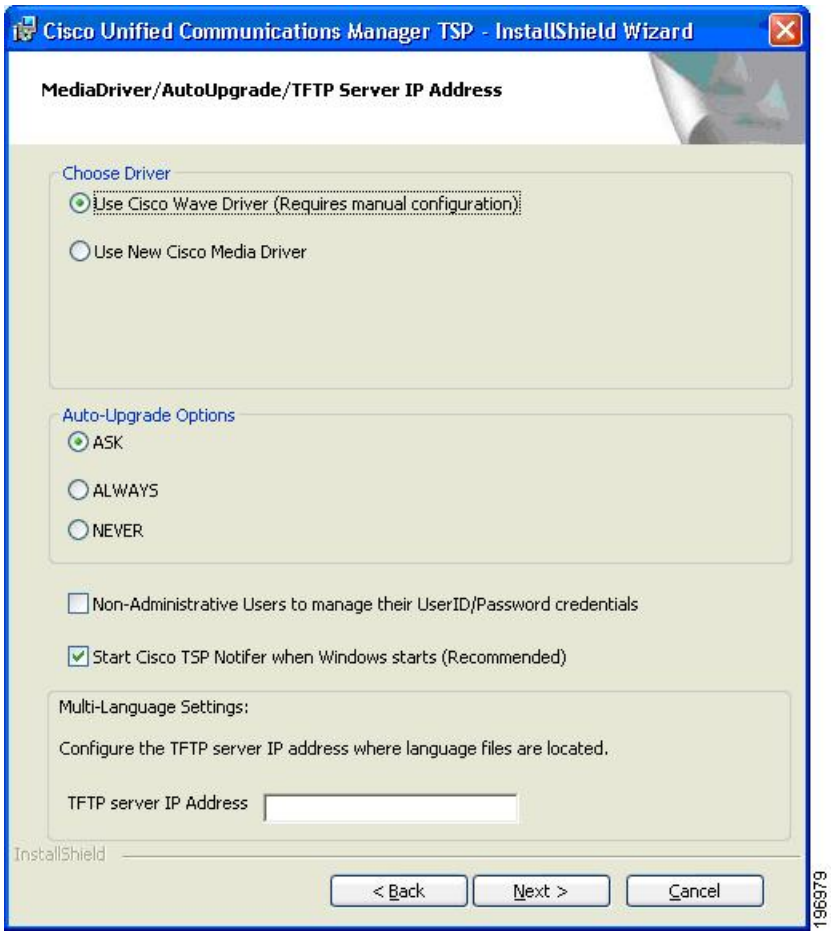
- Unified CM TSP Authentication failed -Check UserID and Password
- Unified CM TSP Initialization failed -User not configured in Cisco Unified CM for CTI usage
- Unified CM TSP Initialization error -Check and update Cisco Unified CM TSP version
- Unified CM TSP Version is incompatible with Cisco Unified CM version

- Unified CM TSP Initialization failed -User not configured in Cisco Unified CM for secure CTI usage
- Unified CM TSP Initialization failed -Cisco Unified CM security configuration does not match with Unified CM TSP
- Unified CM TSP Initialization failed -Invalid security certificate
- Unified CM TSP Initialization failed -Security certificate compromised

Multi-Language Settings

Populate the TFTP server where locale files have been installed. The locale files are downloaded after the installation is completed and the PC rebooted. Use the Cisco TAPI Configuration tool to set the desired locale.

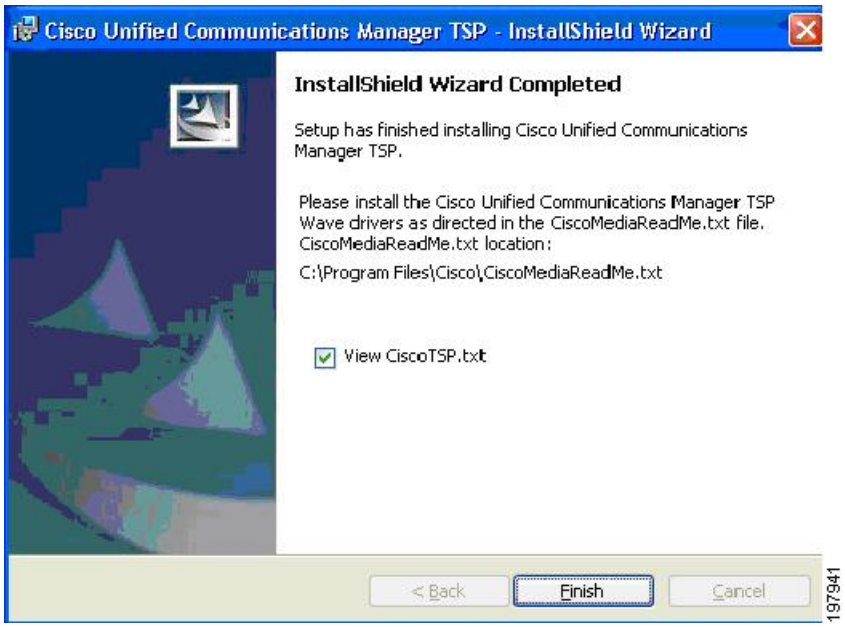
Figure 14: Media Driver/AutoUpgrade/TFTP Server IP Address Screen



Installation Completed

The following screen displays when the installation is complete. You must reboot your computer after the installation. You can refer to the Release Notes for further details.

Figure 15: InstallShield Wizard Completed Screen

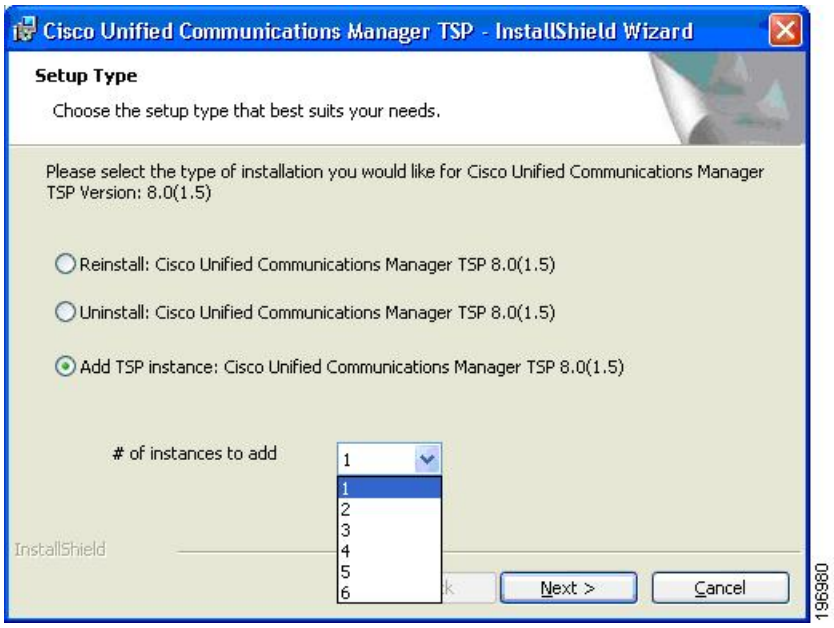


Reinstall or Add a New Instance

If a previous version of the Cisco TSP client is detected and the version of the existing client matches the installer, the Setup Type screen, shown below, displays with the following options:

- Reinstall—Select this option to reinstall the TSP client. This option will be available only if the same version of the TSP client is detected.
- Uninstall—Select this option to remove the Cisco TSP from the PC.
- Add TSP Instance—Select this option to add additional TSP instances. The drop-down menu controls the number of instances to add. The number of instances is limited to 10, so the number of instances that can be added is limited to the maximum number minus the number of instances already installed. If additional instances are added, the installer prompts the Configure TSP Instance for all new instances.

Figure 16: Setup Type Screen

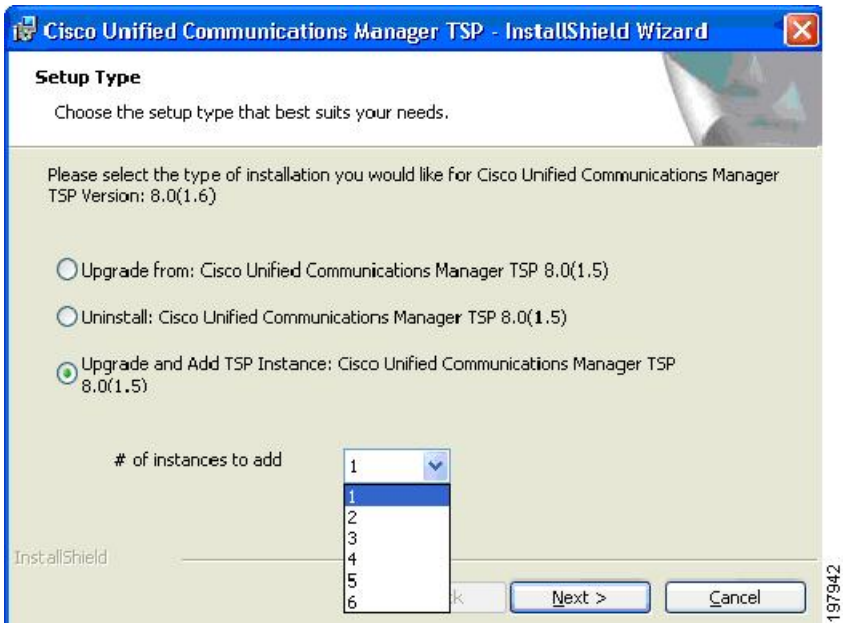


Upgrading CiscoTSP

If a previous version of the Cisco TSP client is detected and the version of the installer is newer than the one already installed, the Setup Type screen, shown below, displays with the following options:

- Upgrade—Select this option to upgrade all existing TSP instances and client.
- Uninstall Cisco TSP—Select this option to remove the Cisco TSP from the PC.
- Upgrade and Add a New TSP Instance—Select this option to add additional TSP instances and upgrade all instances to the newer version. The drop-down menu controls the number of instances to add. The number of instances is limited to 10, so the number of instances that can be added is limited to the maximum number minus the number of instances already installed. If additional instances are added, the installer will prompt Configure TSP Instance for all new instances.

Figure 17: Setup Type Screen

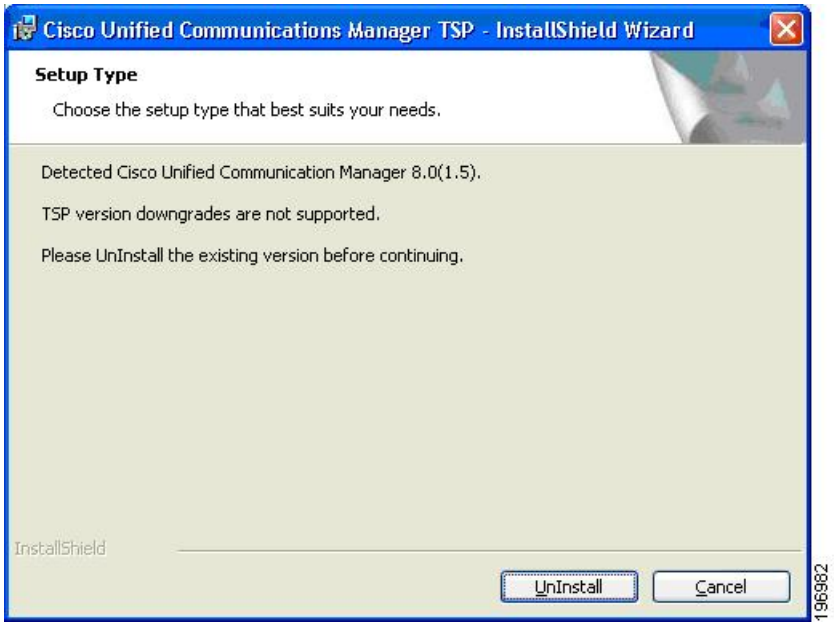


Downgrade or Uninstall of Cisco TSP

If a previous version of the Cisco TSP client is detected and the version of the installer is older than the one already installed, the Setup Type screen, shown below, displays with this option:

- **Uninstall**—Select this option to remove the Cisco TSP from your computer.

Figure 18: Setup Type Screen - Uninstall Option



Silent Installation of Cisco Unified CM TSP

Cisco TSP 8.0 introduces a new Silent Installation feature that allows the Cisco TSP to be remotely installed using Microsoft Group Policy or other remote installation tools. Refer to the list of silent installation parameters to determine the correct settings based on the desired configuration:

Example 1

New Cisco TSP installation which does not require Cisco Media Driver or Cisco Wave Driver:

USER ID = bob

PASSWORD = cisco123

CTIManager1 = 1.1.1.1

CTI1_TYPE = IPV4

```
Cisco TSP.exe /s /v"/qn PASS = \"cisco123\" USER = \"bob\" CTI1 = \"1.1.1.1\"
CTI1_TYPE = \"ipv4\""
```

Example 2

New Cisco TSP installation which requires the Cisco Media Driver:

USER ID = bob

PASSWORD = cisco123

CTIManager1 = 1.1.1.1

CTI1_TYPE = IPV4

DRIVER_TYPE = NEW

MDP_START = 30000

MDP_END = 31000

```
Cisco TSP.exe /s /v"/qn PASS = \"cisco123\" USER = \"bob\" CTI1 = \"1.1.1.1\"
CTI1_TYPE = \"ipv4\" DRIVER_TYPE = \"NEW\" MDP_START = \"30000\" MDP_END =
\"31000\""
```

Syntax Format

- No spaces between the parameter and the " = " sign
- "\" is used as an escape character, so \" is needed to indicate a double quote
- If no parameters are used:
    - For Silent Install -CiscoTSP.exe /s /v/"qn"
    - Silent Upgrade -CiscoTSP.exe /s /v"/qn [REINSTALL = \"ALL\" REBOOT = \"ReallySuppress\"]"

- Silent Reinstall -CiscoTSP.exe /s /v"/qn REINSTALL = \"ALL\" REBOOT = \"ReallySuppress\""

See the following table for the list of parameters that can be passed (default):

Table 3: Silent Install Parameters

Name	Comment	Valid values
USER	User name used to during provider initializing	All
PASS	Password used to during provider initializing	All
CTI1	CTIManager 1 IP address	All
CTI2	CTIManager 2 IP address	All
NONAD	Non-admin can modify the userid/password	"NO", "YES"
CTI1	Ipv4, Ipv6, or hostname	"NONE", "IPV4", "IPV6", "HOST"
CTI2_TYPE	None, Ipv4, Ipv6, or hostname	"NONE", "IPV4", "IPV6", "HOST"
AUTOUPGRADE_TYPE	Ask, Always, Never	"ASK", "NEVER", "ALWAYS"
DRIVER_TYPE	Cisco Wave Driver, Cisco Media Driver	"WAVE", "MEDIA"
NOTIFIER	Start the TSP Notifier automatically during login	"NO", "YES"
NUM_WDP	Number of Legacy Wave Driver Ports	0 -255
MDP_START	Media Driver Port Start Range	0 -65535
MDP_END	Media Driver Port End Range	0 -65535

Upgrading Unified CM TSP Client to Release 8.5(1) Using Silent Installation

After the silent fresh installation or silent upgrade from previous versions of TSP to the 8.5(1) client, the system must be rebooted for changes to take effect.

Note In a case where more than one TSP instance is installed, the client UI-based installation package must be used when upgrading the TSP to maintain the existing configuration settings.

Using Cisco TSP

The following section describes program group and program elements.

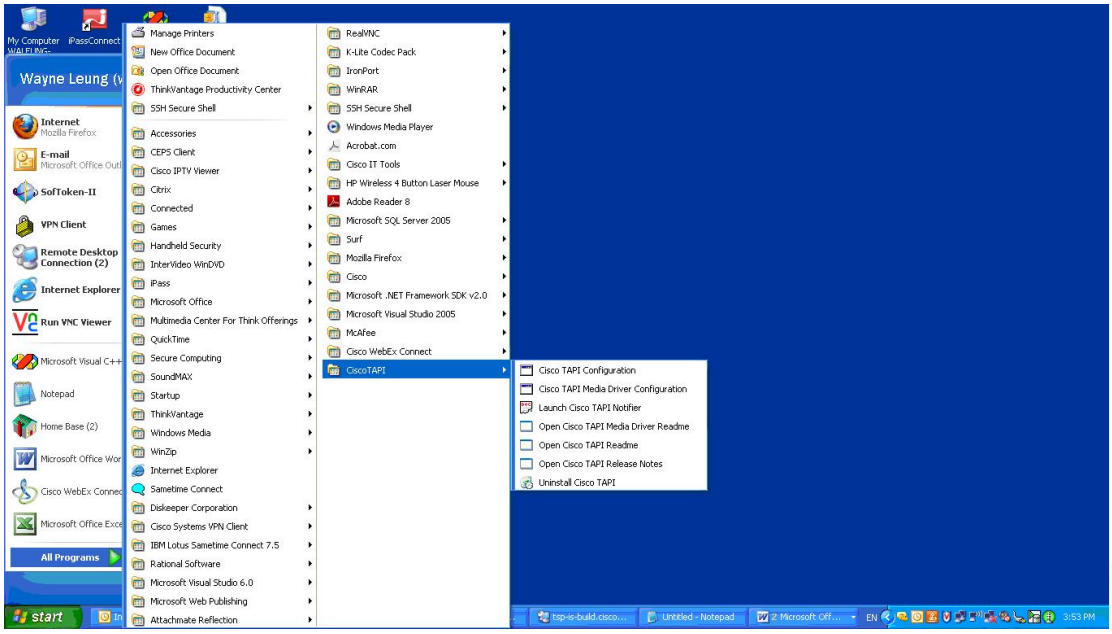
Program Group and Program Elements

There is a new program group called Cisco TAPI created during installation and contains the following program elements:

- Cisco TAPI Configuration—Displays the configuration tool for all TSP instances. If the logged-in User has local Administrative rights, then all settings can be changed. If the administrator selects **Allow Standard User to Update Credentials**, then the Cisco TAPI Configuration tool only allows the UserID/Password to be updated.
- Cisco TAPI Media Driver Configuration—Displays the configuration settings for the Cisco Media Driver. The settings apply to all configured TSP instances.
- Launch Cisco TAPI Notifier—Starts the Cisco TSP Notifier tool to help detect communication issues between Cisco Unified CM and the PC.
- Open Cisco TAPI Media Driver Readme—Displays the Cisco Media Driver Readme file.
- Open Cisco TAPI Readme—Displays the Cisco TSP Readme file.
- Open Cisco TAPI Release Notes—Displays the Cisco TSP Release Notes for the installed version.
- Uninstall Cisco TAPI—Removes Cisco TSP and Cisco Media Driver from the PC. The Cisco Wave Driver (if installed) must be removed using the Telephony Services applet found in the Control Panel.

After installation is completed, the program group appears as shown below.

Figure 19: Cisco TAPI Menu From MS Windows

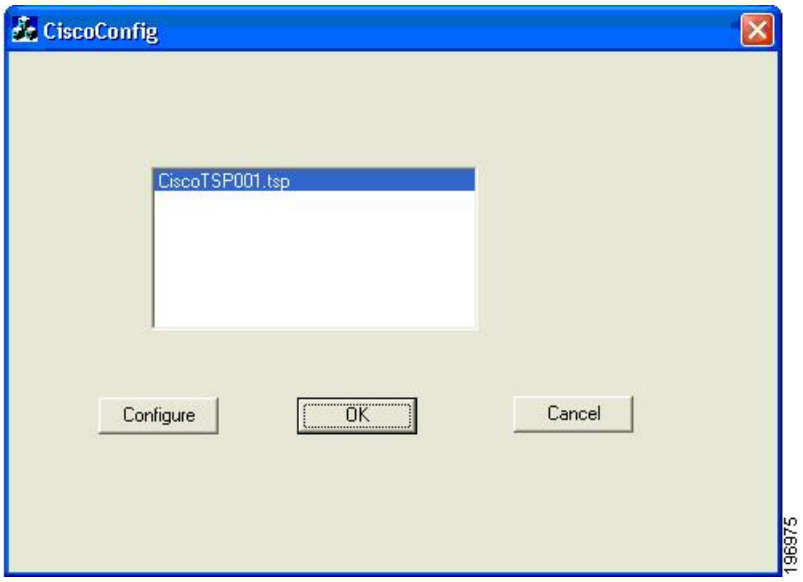


Invoking the uninstall of Cisco TAPI the removes both the TSP and the next generation Media Driver (the old existing Wave Driver is NOT removed). It removes the Program Group from the Windows Task Bar along with all the program elements.

Modifying Cisco TSP Configuration

Select Cisco TAPI Configuration from the Cisco TSP Program Group as shown below in the CiscoConfig dialog.

Figure 20: CiscoConfig Dialog



Cisco Unified CM TSP Configuration Settings

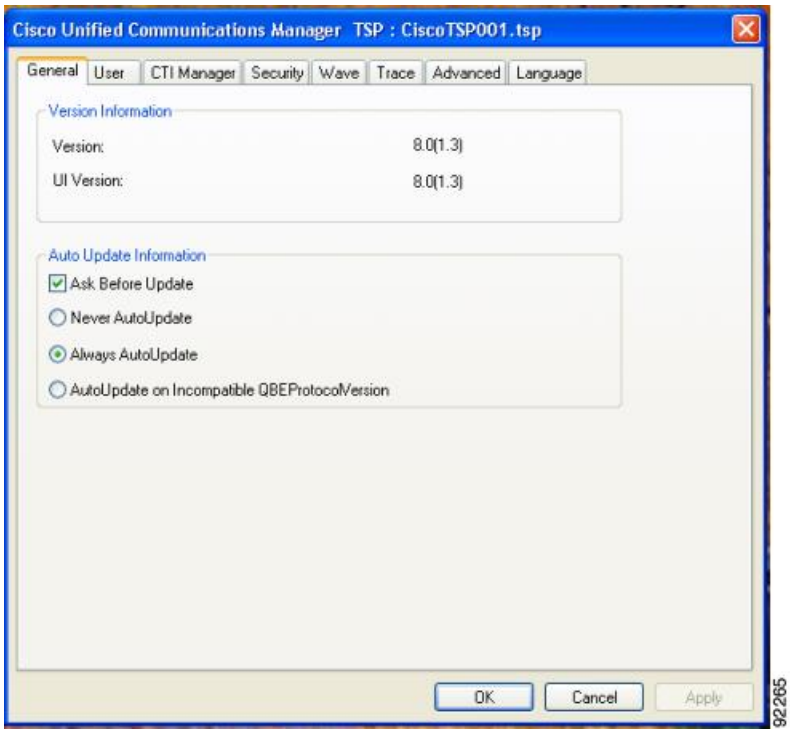
The following sections describe the tabs in the Cisco Unified CM TSP dialog box:

- General, on page 120
- User, on page 121
- CTI Manager, on page 122
- Security, on page 124
- Configuring Cisco Media Driver and Cisco Wave Driver, on page 126
- Trace, on page 129
- Advanced, on page 130
- Language, on page 132

General

The General Tab displays the Cisco TSP version and auto-update settings, as illustrated below.

Figure 21: Cisco Unified Communications Manager TSP General Tab



The following table lists the General tab fields that must be set and their descriptions.

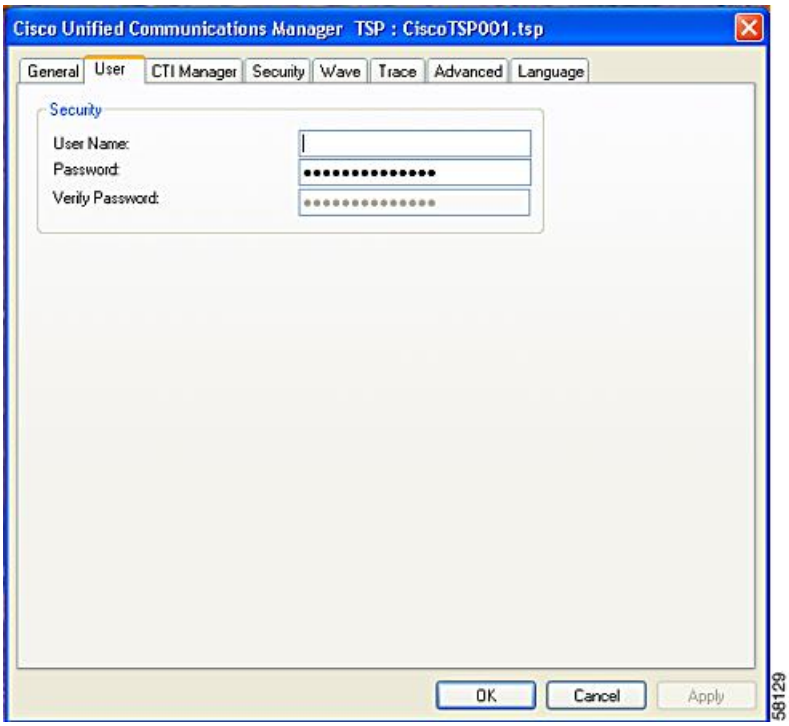
Table 4: Auto Update Information Fields

Field	Description
Ask Before Update	Auto-upgrade service prompts the User to upgrade the TSP client.
Never AutoUpdate	Select this to disable the auto-update service.
Always AutoUpdate	Select this to enable automatic updates when new TSP client versions are detected.
AutoUpdate on Incompatible QBEProtocolVersion	Select this to allow the Cisco TSP to auto update only when the local TSP version is incompatible with the Unified CM version.

User

The User tab allows you to set the user name and password, as illustrated in below.

Figure 22: Cisco Unified Communications Manager TSP User Tab



The table below describes the fields for the User tab that must be set.

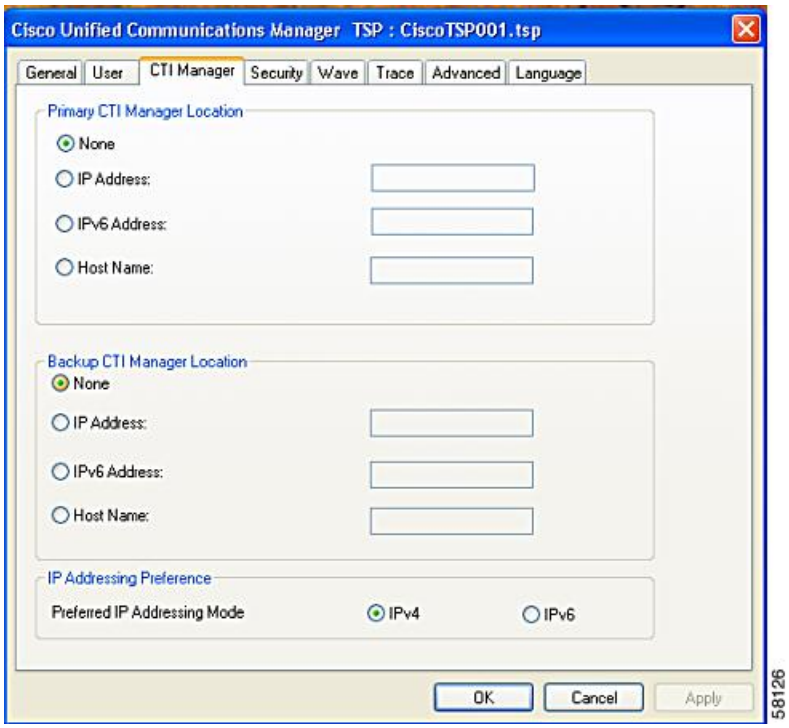
Table 5: User Tab Configuration Fields

Field	Description
User Name	Enter the user name of the user. The TSP instance will access devices and lines that are associated with this User in Cisco Unified CM. Make sure that this User is enabled for CTI using the Cisco Unified CM Administration UI **Note** You can designate only one user name and password to be active at any time for any one TSP instance.
Password	Enter the password that is associated with the user. The computer encrypts the password and stores it in the registry.
Verify Password	Reenter the user password.

CTI Manager

The CTI Manager tab allows you to configure primary and secondary CTI Manager information, as illustrated in below.

Figure 23: Cisco Unified Communications Manager TSP CTI Manager Tab



The table below describes the CTI Manager tab fields that must be set.

Table 6: CTI Manager Configuration Fields

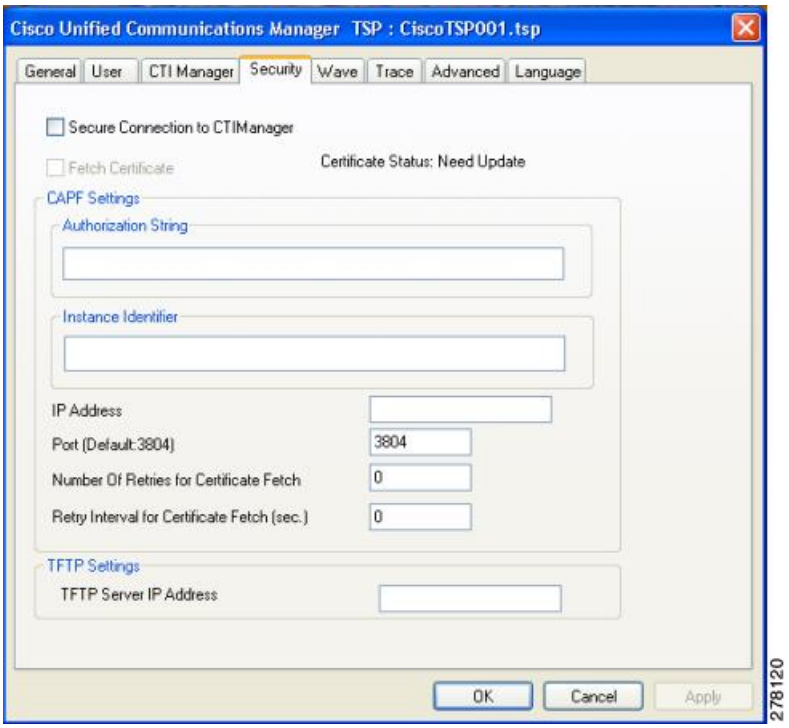
Field	Description
Primary CTI Manager Location	Use this field to specify the CTI Manager to which the TSP attempts to connect to first. Select any of these options: • Select IPv4 Address radio button and use the text box to provide the IPv4 address of the Primary CTI Manager, or • Use IPv6 Address radio button and use the text box to provide the IPv6 address, if application needs to connect using IPv6 with Primary CTIManager, or • Select the Host Name radio button and enter the host name of Primary CTI Manager.
Backup CTI Manager Location	Use this field to specify the CTI Manager to which the TSP attempts to connect to first. Select any of these options: • Select the IPv4 Address radio button and use the text box to provide the IPv4 address of the Backup CTI Manager, or • Select IPv6 Address radio button and use the text box to provide the IPv6 address of the Backup CTI Manager, or • Select the Host Name radio button and enter the host name of the Backup CTI Manager.

Field	Description
IP Addressing Preference	Preferred addressing mode with which the application tries to connect with the CTIManager when IPv4 or IPv6 address is available. If the TSP fails to connect, it will retry with the other IP addressing modes.

Security

The Security tab allow you to configure the security settings for the selected TSP instance, as illustrated in below.

Figure 24: Cisco Unified Communications Manager TSP Security Tab



The table below describes the Security tab fields that must be set.

Table 7: Security Tab Configuration Fields

Field	Description
Secure Connection to CTIManager	If selected, TSP will secure the connection with CTIManager. Default setting is a non-secure connection, so the setting is unchecked. It is important that the security flag for the TSP User must be enabled through the Cisco Unified CM Administration UI. CTIManager performs a verification to check whether the User connecting via TLS is allowed to have secure access. CTIManager allows only security-enabled users to connect using TLS. The User flag to enable security requires the cluster security to also be enabled/set, otherwise the connection has to be non-secure.
Fetch Certificates	Select this option for the Cisco TSP to download certificate files if they are not already available or installed. This is performed when the certificate status is Need Update. This setting is not stored anywhere and is used only to update the Client certificate when checked and is cleared automatically.
Authorization String	Provide the authentication (authorization) string generated under the CAPF profile. To install or upgrade a locally stored certificate, the User must enter the authentication (authorization) string. This string supports one-time use only; after you use the string for an instance, you cannot use it again. This is required for Client authentication with CAPF Server and Private Key storage on client machine.
Instance Identifier	Provide Instance ID for CAPF end User profile: Each secure connection to CTIManager must have its own certificate for authentication. With the restriction of a distinct certificate per connection, CAPF Server must verify that the user with appropriate AuthCode and InstanceID is requesting the certificate. CAPF server uses the AuthCode and InstanceID to verify the User's identity. Once CAPF server provides a certificate it clears the AuthCode to ensure that only one instance of an application requests a certificate based on a single AuthCode. Cisco Unified CM Administration UI allows User configuration to provide multiple InstanceID and AuthCode.
IP Address	Provide the CAPF server IP address from which to fetch the client certificate.
Port	Provide the CAPF Server Port to connect for Certificate download (Default :3804).

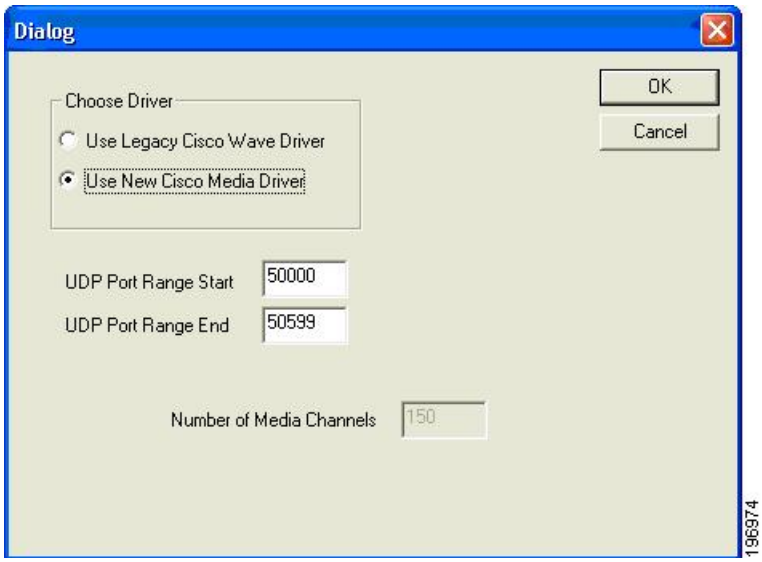
Field	Description
Number of Retries for Certificate Fetch	Indicates the number of retries TSP will perform to connect to the CAPF Server for certificate download in case of an error. This value is used when a communication failure occurs while the certificate installation is taking place. Default value is 0 and range is 0 to 3.
Retry Interval for Certificate Fetch	Indicates the number of seconds the TSP should wait between re-attempting to retrieve the certificate. Default value is 0 and range is 0 to 15.
TFTP IP Address	Indicates the TFTP server IP address from which to fetch the CTL file. CTL file is required to verify the server certificate, sent while mutually authenticating the TLS connection.

Configuring Cisco Media Driver and Cisco Wave Driver

Cisco Media Driver

The Cisco Media Driver configuration can be changed by selecting Configure Cisco TAPI Media Driver from the Cisco TSP Program Group. The user can choose to use the Cisco Wave Driver or the new Cisco Media Driver, as show below. When the Cisco Media Driver is selected, the Udp Port Range Start and End settings appear. The media driver settings are used by all TSP instances. If the Cisco Wave Driver is desired, select Cisco Wave Driver and refer to Cisco Media Driver Selection, on page 109.

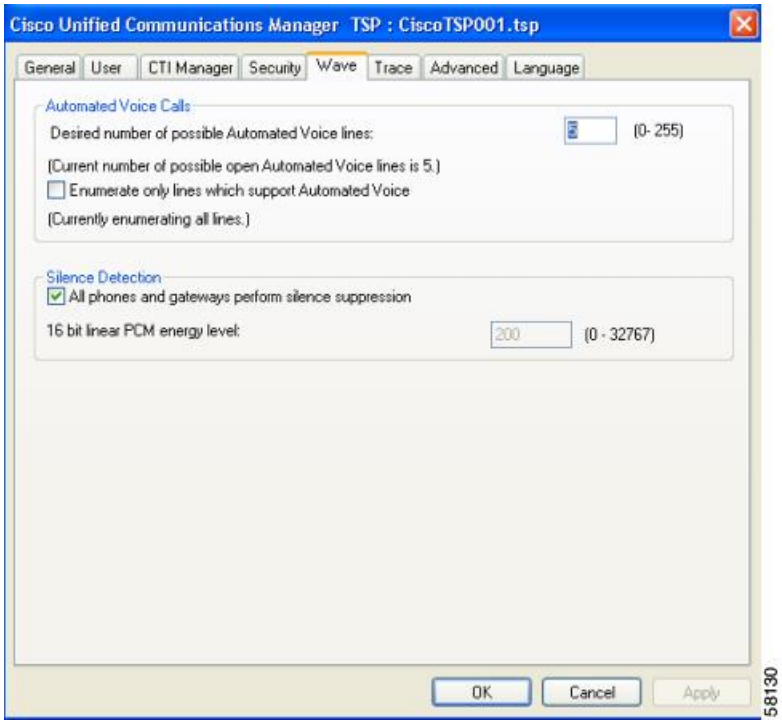
Figure 25: Driver and UDP Port Setting



Cisco Wave Driver

The Cisco Wave Media configuration can be changed by selecting Cisco TAPI Configuration from the Cisco TSP Program Group. Select the desired Instance and click Configure. Choose the Wave tab, see the following figure, to configure the Wave driver settings for the specified instance.

Figure 26: Cisco Unified Communications Manager TSP Wave Tab



The table below describes the Wave tab fields that must be set.

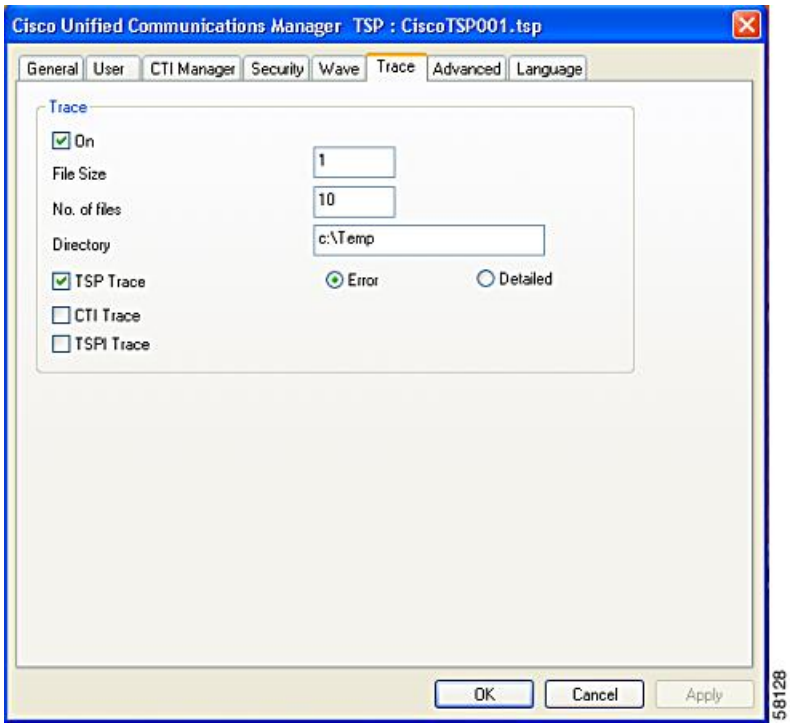
Table 8: Wave Tab Configuration Fields

Field	Description
Automated Voice Calls	The number of Cisco Wave devices determines the number of automated voice lines. (The default value is 5.) The application can open as many CTI ports as the number of Cisco Wave devices that are configured. For example, if you enter "5," you need to create five CTI port devices in Cisco Unified Communications Manager. If you change this number, you need to remove and then reinstall any Cisco Wave devices which were installed. A maximum of 255 Wave devices for all installed TSP instances can be configured. The Wave device limit is specific to Microsoft TAPI which limits the number of Wave devices per Wave driver to 255. The Cisco Media Driver does not have this limitation. When you configure 256 or more Wave devices (including Cisco or other Wave devices), Windows displays the following message when you access the Sounds and Multimedia control panel: "An Error occurred while Windows was working with the Control Panel file C:\Winnt\System32\MMSYS.CPL." The current number of possible automated voice lines designates the maximum number of lines that can be simultaneously opened by using both LINEMEDIAMODE_AUTOMATEDVOICE and LINEMEDIAMODE_INTERACTIVEVOICE. If you are not developing a third-party call control application, check the Enumerate only lines that support automated voice check box, so the Cisco TSP detects only lines that are associated with a CTI port device.
Silence Detection	If you use silence detection, this check box notifies the Wave driver to detect silence on lines that support automated voice calls that are using the Cisco Wave Driver. If the check box is selected (default), the Wave driver searches for the absence of audio-stream RTP packets. As all devices on the network suppress silence and stop sending packets, this method provides a very efficient way for the Wave driver to detect silence. However, if some phones or gateways do not perform silence suppression, the Wave driver must analyze the content of the media stream and, at some threshold, declare that silence is in effect. This CPU-intensive method handles media streams from any type of device. If some phones or gateways on your network do not perform silence suppression, you must specify the energy level at which the Wave driver declares that silence is in effect. This value of the 16-bit linear energy level ranges from 0 to 32767, and the default value is 200. If all phones and gateways perform silence suppression, the system ignores this value.

Trace

The Trace tab allows you to configure various trace settings, as illustrated below. Changes to trace parameters take effect immediately and do not require a computer reboot, even if the TSP is running.

Figure 27: Cisco Unified Communications Manager TSP Trace Tab



The table below describes the Trace tab fields that must be set.

Table 9: Trace Tab Configuration Fields

Field	Description
On	This setting allows you to enable Global Cisco TSP trace. Select the check box to enable Cisco TSP trace. When you enable trace, you can modify other trace parameters in the dialog box. The Cisco TSP trace depends on the values that you enter in these fields. Clear the check box to disable Cisco TSP trace. When you disable trace, you cannot choose any trace parameters in the dialog box, and TSP ignores the values that are entered in these fields.
File size	Default file size is 1 MB.
No. of files	Use this field to specify the maximum number of trace files. The default value is 10. File numbering occurs in a rotating sequence starting at 0. The counter restarts at 0 after it reaches the maximum number of files minus one.

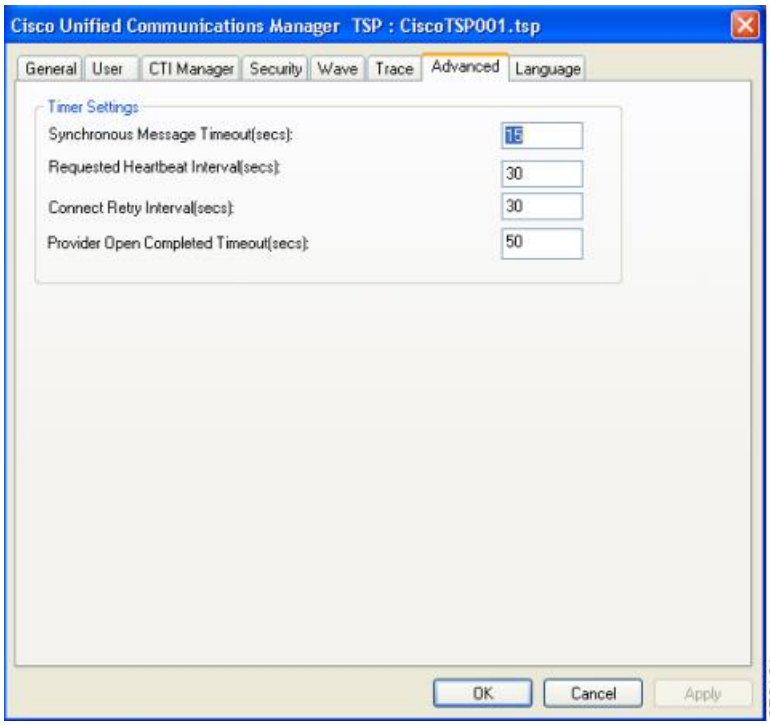
Field	Description
Directory	Use this field to specify the location in which trace files for all CiscoTSPs are stored. The system creates a subdirectory for each instance of CiscoTSP. For example, the CiscoTSP001Log directory stores Cisco TSP 1 log files. The system creates trace files with filename TSP001Debug000xxx.txt for each TSP in its respective subdirectory.
TSP Trace	This setting activates internal TSP tracing. When you activate TSP tracing, Cisco Unified TSP logs internal debug information that you can use for debugging purposes. You can choose one of the following levels: Error—Logs only TSP errors. Detailed—Logs all TSP details (such as log function calls in the order that they are called). The system checks the TSP Trace check box and chooses the Error radio button by default.
CTI Trace	This setting traces messages that flow between Cisco TSP and CTIManager. By default, CTI Trace is not selected.
TSPI Trace	This setting traces all messages and function calls between Microsoft TAPI and the Cisco TSP. By default, TSPI Trace is not selected. If TSPI Trace is enabled, Cisco TSP traces all the function calls that Microsoft TAPI makes to the Cisco TSP with parameters and messages (events) from Cisco TSP to MS TAPI.

Advanced

The Advanced tab allows you to configure timer settings, as illustrated in below.

Note Timer settings should only be changed when necessary and recommended by Cisco Technical Assistance Center (TAC).

Figure 28: Cisco Unified Communications Manager TSP Advanced Tab



The table below describes the Advanced tab fields that must be set.

Table 10: Advanced Configuration Fields

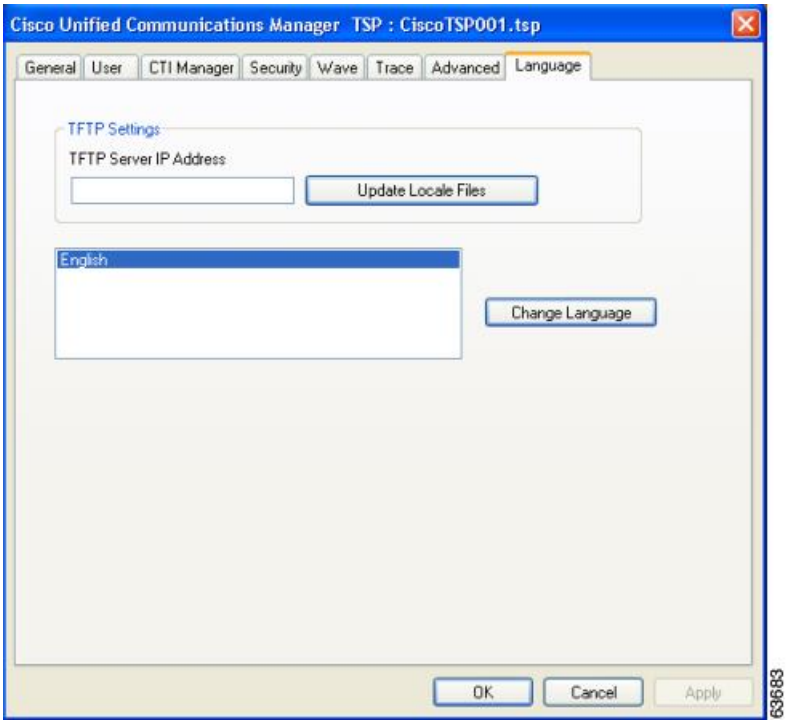
Field	Description
Synchronous Message Timeout (secs)	Use this field to designate the time that the TSP waits to receive a response to a synchronous message. The value displays in seconds, and the default value is 15. Range goes from 5 to 60 seconds.
Requested Heartbeat Interval (secs)	Use this field to designate the interval at which the heartbeat messages are sent from TSP to detect whether the CTI Manager connection is still alive. TSP sends heartbeats when no traffic exists between the TSP and CTI Manager for 30 seconds or more. The default interval is 30 seconds. Range goes from 30 to 300 seconds.
Connect Retry Interval (secs)	Use this field to designate the interval between reconnection attempts after a CTI Manager connection failure. The default value is 30 seconds. Range goes from 15 to 300 seconds.

Field	Description
Provider Open Completed Timeout (secs)	Use this field to designate the time that Cisco Unified TSP waits for a Provider Open Completed Event, which indicates the CTI Manager is initialized and ready to serve TSP requests. Be aware that CTI initialization time is directly proportional to the number of devices that are configured in the system. The default value is 50 seconds. Range goes from 5 to 900 seconds.

Language

The Language tab displays the installed locales and allows for localization of the client user interface, as illustrated in below.

Figure 29: Cisco Unified Communications Manager TSP Language Tab



The following table desctribes the Language tab fields.

Table 11: Language Configuration Fields

Field	Description
TFTP Server IP Address	Configure the TFTP server IP where the COP files are installed for the desired Locales.
Update Locale Files	Downloads the locale files from the configured TFTP server and extracts those files to the resources directory in the client machine

Field	Description
Change Language	Choose a language and click this to reload the tabs with the text in that language.

Verify the Cisco Unified CM TSP Installation

Use the Microsoft Windows Phone Dialer Application to verify that the Cisco TSP client has been installed and is configured correctly. Locate the dialer application by performing a search for dialer.exe.

Procedure

Step 1 Open the Dialer application by locating it in Windows Explorer and double-clicking it

Step 2 Choose **Edit** > **Options**.

Step 3 Choose Phone as the **Preferred Line for Calling**.

Step 4 In the Line Used For area, choose one Cisco Line in the **Phone Calls** drop-down menu.

Step 5 Click **OK**.

Step 6 Click **Dial**.

Step 7 Enter a number to dial, choose Phone Call in the Dial as box, and then click **Place Call**.

What to do next

If the call is successful, the Cisco TSP client is installed correctly.

If you encounter problems during this procedure, or if no lines appear in the line drop-down list on the dialer application, check the following items:

- Verify the Cisco TSP configuration settings by opening the Cisco TAPI Configuration tool and verifying the configured parameters.
- Reboot the computer to ensure all configuration options and installation processes have completed and are updated successfully.
- Test the network link between the Cisco TSP client machine and Cisco Unified CM by using the ping command to check connectivity.
- Ensure that the Cisco Unified CM CTIManager is running.

Managing the Cisco Unified CM TSP

You can perform the following actions on all installed TSPs:

- Reinstall the existing Cisco Unified TSP client (same version)
- Upgrade to the newer version of the Cisco TSP client

- Remove Cisco TSP from the Telephone Service Provider List
- Uninstall the Cisco TSP client
- Uninstall the Cisco Wave Driver

Related Topics

Reinstall the Cisco Unified TSP, on page 134
Upgrade the Cisco Unified TSP, on page 134
Remove Cisco Unified TSP From the Provider List, on page 135
Uninstall the Cisco TSP Client, on page 135
Uninstall the Cisco Wave Driver, on page 135
Auto Update for Cisco Unified TSP Upgrades, on page 136

Reinstall the Cisco Unified TSP

Use the following procedure to reinstall the Cisco Unified TSP on all supported platforms.

Procedure

Step 1 Open the Control Panel and double-click **Add/Remove Programs**.

Step 2 Choose Cisco Unified TSP and click **Change**.

The Cisco Unified TSP maintenance install dialog box displays.

Step 3 Click the **Reinstall TSP** radio button and click **Next**.

Step 4 Follow the online instructions.

Step 5 Restart the computer

Upgrade the Cisco Unified TSP

Use the following procedure to upgrade the Cisco Unified TSP on all supported platforms.

Procedure

Step 1 Download and save the new TSP client on the target PC and double-click the installer.

Step 2 Select the **Upgrade from TSP** radio button and click **Next**.

Step 3 Follow the online instructions.

Step 4 Restart the computer

Remove Cisco Unified TSP From the Provider List

This process removes the Cisco TSP from the Microsoft provider list but does not uninstall the TSP client. To make these changes, perform the following steps.

Procedure

Step 1 Open the Control Panel.
Step 2 Double-click the **Phone and Modem** icon.
Step 3 Click the **Advanced** tab.
Step 4 Choose the Cisco Unified TSP that you want to remove.
Step 5 To delete the Cisco Unified TSP from the list, click **Remove**.

Uninstall the Cisco TSP Client

To remove the Cisco TSP client, choose Uninstall Cisco TAPI from the Cisco TSP Program Group.

Alternatively, use the following procedure to uninstall the Cisco TSP on all supported platforms.

Procedure

Step 1 Open the Control Panel and double-click **Add/Remove Programs**.
Step 2 Choose the Cisco Unified TSP that you want to remove and click **Remove**.

The Cisco TSP maintenance install dialog box displays.

Step 3 Select the **Uninstall: Remove the installed TSP** radio button and click **Next**.
Step 4 Follow the online instructions.
Step 5 Restart the computer

Uninstall the Cisco Wave Driver

To remove the Cisco Wave Driver, perform one of the following procedures.

Uninstall the Cisco Wave Driver for Windows 2003

To remove the Cisco Wave Driver for Windows 2003, perform the following steps.

Procedure

Step 1 Open the Control Panel.
Step 2 Select **Sound and Audio Devices**.

Step 3 Click the **Hardware** tab.

Step 4 Select **Cisco Unified Communications Manager TSP Wave Driver**.

Step 5 Click **Properties.**

Step 6 Click the **Driver** tab.

Step 7 Click **Uninstall** and **OK** to remove.

Step 8 If the Cisco TAPI Wave Driver entry is still displayed, close and open the window again to verify that it has been removed.

Step 9 Restart the computer.

Uninstall the Cisco Wave Driver for Windows 2008

To remove the Cisco Wave Driver for Windows 2008, perform the following steps.

Procedure

Step 1 Open the Control Panel.

Step 2 Select **Device Manager**.

Step 3 Click the **Sounds, Video, and Game Controllers** tab.

Step 4 Select **Cisco Unified Communications Manager TSP Wave Driver**.

Step 5 Right click and select **Uninstall**.

Step 6 Restart the computer.

Auto Update for Cisco Unified TSP Upgrades

Cisco TSP supports an auto update feature, so the latest client is downloaded and installed on the client machine automatically. When Cisco Unified Communications Manager is upgraded to a higher version and the Cisco TSP client Auto-Upgrade option is set to Ask or Always, the latest Cisco TSP client will be downloaded to the computer automatically. If Ask is configured, the user will be prompted to upgrade the client. If Always is configured, the client will upgrade automatically. The logged-in user must have local Administrative rights to install applications to use the auto-upgrade feature.

Auto Update Behavior

As part of the Cisco TSP initialization when the application issues lineInitializeEx, Cisco TSP queries the current TSP client version information that is available from the Cisco Unified CM server running CTIManager. Cisco TSP compares the installed Cisco TSP version with the client version available on the server. If a newer version is available and Auto-Upgrade is enabled, the Cisco TSP triggers the auto-update process. As part of Auto-Upgrade, Cisco TSP behaves in the following ways on different platforms.

After Cisco TSP detects that an upgradeable version is available, Cisco TSP reports 0 lines to the application and removes the Cisco TSP provider from the provider list. If a new TSP client version is detected during the reconnect time, the running applications receive LINE_REMOVE for all lines, which are already initialized and are in an OutOfService state. Cisco TSP silently upgrades to the new version that was downloaded from

Cisco Unified CM and puts the Cisco TSP provider back on the provider list. All the running applications receive LINE_CREATE messages.

Windows XP supports multiple user logon sessions (as part of fast user switching), however, the system supports Auto Upgrade only for the first logged-on User. If multiple User sessions are active, Cisco TSP only supports the Auto Upgrade functionality for the first logged-on user.

Note If a User has multiple Cisco TSPs installed on the client machine, the system enables only the first Cisco TSP instance to set up the Auto Upgrade configuration. All Cisco TSP clients are upgraded to a common version upon version mismatch. From Control Panel, select **Phone & Modem Options**>**Advanced**>**Cisco TSP001**, the General window displays the options for Auto Upgrade.

The Cisco TSP client plug-in location can be changed to a different machine other than the Cisco Unified CM server (if desired). The default location is //<Cisco Unified CM Server IP address or hostname>//ccmpluginsserver.

If Silent upgrade fails on any listed platforms for any reason, the old Cisco TSP provider(s) do not get put back on the provider list to avoid any looping of the Auto Upgrade process. Ensure that the update options are cleared and the providers added to provider list manually. Update the Cisco TSP manually or fix the issue(s) encountered during Auto Upgrade and reinitialize the Cisco TSP client to re-trigger the Auto Upgrade process.

Note TSPAutoinstall.exe requires the Telephony Service LocalSystem logon option to Allow Service to interact with Desktop. If the logon option is not set as LocalSystem or logon option is LocalSystem but Allow Service to interact with Desktop is disabled (not selected), then Cisco TSP cannot launch the AutoInstall UI windows and will not succeed.

Note In the 8.5(1) release, the above services are not enabled as TSPAutoInstall.exe runs as an independent background process. Following the installation, upgrade, or reinstall of Cisco TSP, you must disable the User Account Control before the reboot.

Ensure that the following logon options are set for the telephony service.

1. Logon as: **LocalSystem.**
2. Enable the check box **Allow Service to interact with Desktop**.

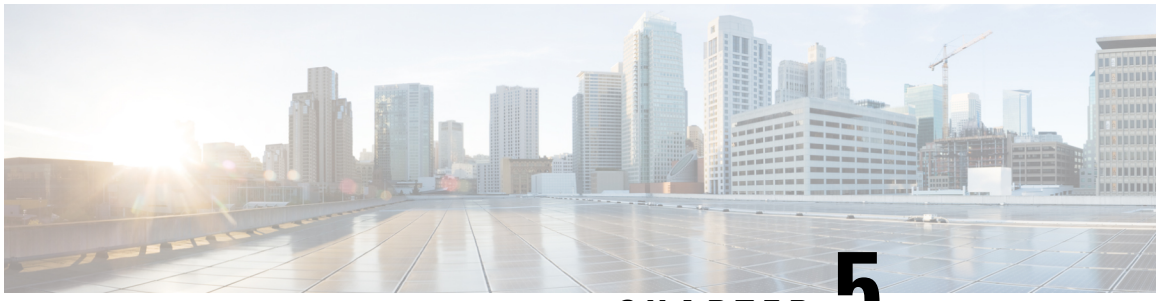
These telephony service settings, when changed, require manual restart of the service to take effect. If the Microsoft Remote Connection Manager service is not disabled, reboot the PC for the changes to take effect.

Cisco TSP Behavior on Windows Upgrade

On upgrade of windows to a higher version, for example from Windows 8 to Windows 10, Cisco TSP must be reinstalled to retain all the previous instances of data present in CiscoTSP Configuration.

Perform the following procedure to reinstall the Cisco Unified TSP.

Reinstall the Cisco Unified TSP, on page 134



CHAPTER 5

Basic TAPI Implementation

This chapter outlines the TAPI 2.1 functions, events, and messages that the Cisco Unified TAPI Service Provider (TSP) supports. This chapter contains functions in the following sections:

Overview

TAPI comprises a set of classes that expose the functionality of the Cisco Unified Communications Solutions. TAPI enables developers to create customized IP telephony applications for Cisco Unified Communications Manager without specific knowledge of the communication protocols between the Cisco Unified Communications Manager and the service provider. For example, a developer could create a TAPI application that communicates with an external voice-messaging system.

TAPI Line Functions

The number of TAPI devices that are configured in the Cisco Unified Communications Manager determines the number of available lines. Cisco Media Driver is used to terminate a media stream in the first-party call control models.

Table 12: TAPI Line Functions Supported

TAPI line functions supported
lineAccept, on page 142
lineAddProvider, on page 142
lineAddToConference, on page 143

TAPI line functions supported
lineAnswer, on page 144
lineBlindTransfer, on page 144
lineCallbackFunc, on page 145
lineClose, on page 146
lineCompleteTransfer, on page 146
lineConfigProvider, on page 147
lineDeallocateCall, on page 148
lineDevSpecific, on page 148
lineDevSpecificFeature, on page 150
lineDial, on page 151
lineDrop, on page 152
lineForward, on page 153
lineGenerateDigits, on page 155
lineGenerateTone, on page 156
lineGetAddressCaps, on page 157
lineGetAddressID, on page 158
lineGetAddressStatus, on page 159
lineGetCallInfo, on page 159
lineGetCallStatus, on page 160
lineGetConfRelatedCalls, on page 160
lineGetDevCaps, on page 161
lineGetID, on page 162
lineGetLineDevStatus, on page 163
lineGetMessage, on page 163
lineGetNewCalls, on page 164
lineGetNumRings, on page 165
lineGetProviderList, on page 166
lineGetRequest, on page 167

TAPI line functions supported
lineGetStatusMessages, on page 168
lineGetTranslateCaps, on page 168
lineHandoff, on page 169
lineHold, on page 170
lineInitialize, on page 171
lineInitializeEx, on page 172
lineMakeCall, on page 173
lineMonitorDigits, on page 174
lineMonitorTones, on page 174
lineNegotiateAPIVersion, on page 175
lineNegotiateExtVersion, on page 176
lineOpen, on page 177
linePark, on page 178
linePrepareAddToConference, on page 179
lineRedirect, on page 181
lineRegisterRequestRecipient, on page 181
lineRemoveFromConference, on page 182
lineSetAppPriority, on page 184
lineSetCallPrivilege, on page 185
lineSetNumRings, on page 186
lineSetStatusMessages, on page 187
lineSetTollList, on page 188
lineSetupConference, on page 189
lineSetupTransfer, on page 190
lineShutdown, on page 190
lineTranslateAddress, on page 191
lineTranslateDialog, on page 192
lineUnhold, on page 194

TAPI line functions supported
lineUnpark, on page 194

lineAccept

The lineAccept function accepts the specified offered call.

Function Details

```
LONG lineAccept(HCALL hCall,
LPCSTR lpsUserUserInfo,
DWORD dwSize
);
```

Parameters

hCall

A handle to the call to be accepted. The application must be an owner of the call. Call state of hCall must be offering.

lpsUserUserInfo

A pointer to a string that contains user-user information to be sent to the remote party as part of the call accept. Leave this pointer NULL if you do not want to send user-user information. User-user information is sent only if supported by the underlying network. The protocol discriminator member for the user-user information, if required, should appear as the first byte of the buffer that is pointed to by lpsUserUserInfo and must be accounted for in dwSize.

Note The Cisco Unified TSP does not support user-user information.

dwSize

The size in bytes of the user-user information in lpsUserUserInfo. If lpsUserUserInfo is NULL, no user-user information gets sent to the calling party, and dwSize is ignored.

lineAddProvider

The lineAddProvider function installs a new telephony service provider into the telephony system.

Function Details

```
LONG WINAPI lineAddProvider(  LPCSTR lpszProviderFilename,
  HWND hwndOwner,
  LPDWORD lpdwPermanentProviderID
);
```

Parameters

lpszProviderFilename

A pointer to a null-terminated string that contains the path of the service provider to be added.

hwndOwner

A handle to a window in which dialog boxes that need to be displayed as part of the installation process (for example, by the service provider's TSPI_providerInstall function) would be attached. Can be NULL to indicate that any window created during the function should have no owner window.

lpdwPermanentProviderID

A pointer to a DWORD-sized memory location into which TAPI writes the permanent provider identifier of the newly installed service provider.

Return Values

Returns zero if request succeeds or a negative number if an error occurs. Possible return values are:

- LINEERR_INIFILECORRUPT
- LINEERR_NOMEM
- LINEERR_INVALPARAM
- LINEERR_NOMULTIPLEINSTANCE
- LINEERR_INVALPOINTER
- LINEERR_OPERATIONFAILED

lineAddToConference

This function takes the consult call that is specified by hConsultCall and adds it to the conference call that is specified by hConfCall.

Function Details

```
LONG lineAddToConference(  HCALL hConfCall,
  HCALL hConsultCall
);
```

Parameters

hConfCall

A pointer to the conference call handle. The state of the conference call must be OnHoldPendingConference or OnHold.

hConsultCall

A pointer to the consult call that will be added to the conference call. The application must be the owner of this call, and it cannot be a member of another conference call. The allowed states of the consult call comprise connected, onHold, proceeding, or ringback

lineAnswer

The lineAnswer function answers the specified offering call.

Note CallProcessing requires previous calls on the device to be in connected call state before answering further calls on the same device. If calls are answered without checking for the call state of previous calls on the same device, then Cisco Unified TSP might return a successful answer response but the call will not go to connected state and needs to be answered again.

Function Details

```
LONG lineAnswer(  HCALL hCall,
  LPCSTR lpsUserUserInfo,
  DWORD dwSize
);
```

Parameters

hCall

A handle to the call to be answered. The application must be an owner of this call. The call state of hCall must be offering or accepted.

lpsUserUserInfo

A pointer to a string that contains user-user information to be sent to the remote party at the time the call is answered. You can leave this pointer NULL if no user-user information will be sent.

User-user information only gets sent if supported by the underlying network. The protocol discriminator field for the user-user information, if required, should be the first byte of the buffer that is pointed to by lpsUserUserInfo and must be accounted for in dwSize.

Note The Cisco Unified TSP does not support user-user information.

dwSize

The size in bytes of the user-user information in lpsUserUserInfo. If lpsUserUserInfo is NULL, no user-user information gets sent to the calling party, and dwSize is ignored.

lineBlindTransfer

The lineBlindTransfer function performs a blind or single-step transfer of the specified call to the specified destination address.

Note The lineBlindTransfer function that is implemented until Cisco Unified TSP 3.3 does not comply with the TAPI specification. This function actually gets implemented as a consultation transfer and not a single-step transfer. From Cisco Unified TSP 4.0, the lineBlindTransfer complies with the TAPI specs wherein the transfer is a single-step transfer.

If the application tries to blind transfer a call to an address that requires a FAC, CMC, or both, then the lineBlindTransfer function will return an error. If a FAC is required, the TSP will return the error LINEERR_FACREQUIRED. If a CMC is required, the TSP will return the error LINEERR_CMCREQUIRED. If both a FAC and a CMC are required, the TSP will return the error LINEERR_FACANDCMCREQUIRED. An application that wants to blind transfer a call to an address that requires a FAC, CMC, or both, should use the lineDevSpecific -BlindTransferFACCMC function.

Function Details

```
LONG lineBlindTransfer(  HCALL hCall,
  LPCSTR lpszDestAddress,
  DWORD dwCountryCode
);
```

Parameters

hCall

A handle to the call to be transferred. The application must be an owner of this call. The call state of hCall must be connected.

lpszDestAddress

A pointer to a NULL-terminated string that identifies the location to which the call is to be transferred. The destination address uses the standard dial number format.

dwCountryCode

The country code of the destination. The implementation uses this parameter to select the call progress protocols for the destination address. If a value of 0 is specified, the defined default call-progress protocol is used.

lineCallbackFunc

The lineCallbackFunc function provides a placeholder for the application-supplied function name.

Function Details

```
VOID FAR PASCAL lineCallbackFunc(  DWORD hDevice,
  DWORD dwMsg,
  DWORD dwCallbackInstance,
  DWORD dwParam1,
  DWORD dwParam2,
  DWORD dwParam3
);
```

Parameters

hDevice

A handle to either a line device or a call that is associated with the callback. The context that dwMsg provides determines the nature of this handle (line handle or call handle). Applications must use the DWORD type for this parameter because using the HANDLE type may generate an error.

dwMsg

A line or call device message.

dwCallbackInstance

Callback instance data that is passed back to the application in the callback. TAPI does not interpret DWORD.

dwParam1

A parameter for the message.

dwParam2

A parameter for the message.

dwParam3

A parameter for the message.

Further Details

For information about parameter values that are passed to this function, see TAPI Line Functions, on page 139.

lineClose

The lineClose function closes the specified open line device.

Function Details

```
LONG lineClose(  HLINE hLine
);
```

Parameter

hLine

A handle to the open line device to be closed. After the line has been successfully closed, this handle no longer remains valid.

lineCompleteTransfer

The lineCompleteTransfer function completes the transfer of the specified call to the party that is connected in the consultation call.

Function Details

```
LONG lineCompleteTransfer(  HCALL hCall,
  HCALL hConsultCall,
  LPHCALL lphConfCall,
  DWORD dwTransferMode
);
```

Parameters

hCall

A handle to the call to be transferred. The application must be an owner of this call. The call state of hCall must be onHold, onHoldPendingTransfer.

hConsultCall

A handle to the call that represents a connection with the destination of the transfer. The application must be comprise an owner of this call. The call state of hConsultCall must be connected, ringback, busy, or proceeding.

lphConfCall

A pointer to a memory location where an hCall handle can be returned. If dwTransferMode is LINETRANSFERMODE_CONFERENCE, the newly created conference call is returned in lphConfCall and the application becomes the sole owner of the conference call. Otherwise, TAPI ignores this parameter.

dwTransferMode

Specifies how the initiated transfer request is to be resolved. This parameter uses the following LINETRANSFERMODE_constant:

- LINETRANSFERMODE_TRANSFER—Resolve the initiated transfer by transferring the initial call to the consultation call.
- LINETRANSFERMODE_CONFERENCE—The transfer gets resolved by establishing a three-way conference among the application, the party connected to the initial call, and the party connected to the consultation call. Selecting this option creates a conference call.

lineConfigProvider

The lineConfigProvider function causes a service provider to display its configuration dialog box. This basically provides a straight pass-through to TSPI_providerConfig.

Function Details

```
LONG WINAPI lineConfigProvider(  HWND hwndOwner,
  DWORD dwPermanentProviderID
);
```

Parameters

hwndOwner

A handle to a window to which the configuration dialog box (displayed by TSPI_providerConfig) is attached. This parameter can equal NULL to indicate that any window that is created during the function should have no owner window.

dwPermanentProviderID

The permanent provider identifier of the service provider to be configured.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INIFILECORRUPT
- LINEERR_NOMEM
- LINEERR_INVALPARAM
- LINEERR_OPERATIONFAILED

lineDeallocateCall

The lineDeallocateCall function deallocates the specified call handle.

Function Details

```
LONG lineDeallocateCall(  HCALL hCall
);
```

Parameter

hCall

The call handle to be deallocated. An application with monitoring privileges for a call can always deallocate its handle for that call. An application with owner privilege for a call can deallocate its handle unless it is the sole owner of the call and the call is not in the idle state. The call handle is invalid after it is deallocated.

lineDevSpecific

The lineDevSpecific function enables service providers to provide access to features that other TAPI functions do not offer. The extensions are device-specific and the applications must be able to read the extensions to take advantage of these extensions.

When used with the Cisco Unified TSP, lineDevSpecific can be used to:

- Enable the message waiting lamp for a particular line.
- Handle the audio stream (instead of using the provided Cisco wave driver).
- Turn On or Off the reporting of media streaming messages for a particular line.
- Register a CTI port or route point for dynamic media termination.

- Set the IP address and the UDP port of a call at a CTI port or route point with dynamic media termination.
- Redirect a Call and Reset the OriginalCalledID of the call to the party that is the destination of the redirect.
- Redirect a call and set the OriginalCalledID of the call to any party.
- Join two or more calls into one conference call.
- Redirect a Call to a destination that requires a FAC, CMC, or both.
- Blind Transfer a Call to a destination that requires a FAC, CMC, or both.
- Open a CTI port in third party mode.
- Set the SRTP algorithm IDs that a CTI port supports.
- Acquire any CTI-controllable device in the Cisco Unified Communications Manager system, which needs to be opened in super provider mode.
- Deacquire any CTI-controllable device in the Cisco Unified Communications Manager system.
- Trigger the actual line open from the TSP side. This is used for the delayed open mechanism.
- Initiate TalkBack on the Intercom Whisper call of the Intercom line
- Query SpeedDial and Label setting of a Intercom line.
- Set SpeedDial and Label setting of a Intercom line.
- Start monitoring a call
- Start recording of a call
- Stop recording of a call
- Direct call with feature priority (see Secure Conference, on page 83 for more information.
- Transfer without media
- Direct Transfer
- Message Summary
- Register call pickup group for notification
- Unregister call pickup group for notification
- Call pickup request
- Start send media to BIB
- Stop send media to BIB
- Agent zip tone
- Enable feature
- Add remote destination
- Remove remote destination
- Update remote destination
- Hold enhancement

Note In Cisco Unified TSP Releases 4.0 and later, the TSP no longer supports the ability to perform a SwapHold/SetupTransfer on two calls on a line in the CONNECTED and the ONHOLD call states. Therefore, these calls can be transferred by using lineCompleteTransfer. Cisco Unified TSP Releases 4.0 and later enable to transfer these calls using the lineCompleteTransfer function without having to perform the SwapHold/SetupTransfer beforehand.

Function Details

```
LONG lineDevSpecific(  HLINE hLine,
  DWORD dwAddressID,
  HCALL hCall,
  LPVOID lpParams,
  DWORD dwSize
);
```

Parameters

hLine

A handle to a line device. This parameter is required.

dwAddressID

An address identifier on the given line device.

hCall

A handle to a call. Although this parameter is optional, if it is specified, the call that it represents must belong to the hLine line device. The call state of hCall is device specific.

lpParams

A pointer to a memory area that is used to hold a parameter block. The format of this parameter block specifies device specific, and TAPI passes its contents to or from the service provider.

dwSize

The size in bytes of the parameter block area.

lineDevSpecificFeature

The lineDevSpecificFeature function enables service providers to provide access to features that other TAPI functions do not offer. The extensions are device-specific and the applications must be able to read the extensions to take advantage of these extensions. When used with the Cisco TSP, lineDevSpecificFeature can be used to enable/disable Do-Not-Disturb feature on a device.

Function Details

```
LONG lineDevSpecificFeature(HLINE hLine,
DWORD dwFeature,
LPVOID lpParams,
DWORD dwSize
);
```

Parameters

hLine

A handle to a line device. This parameter is required.

dwFeature

Feature to invoke on the line device. This parameter uses the PHONEBUTTONFUNCTION_TAPI constants. When used with the Cisco TSP, the only value that is considered valid is PHONEBUTTONFUNCTION_DONOTDISTURB (0x0000001A).

lpParams

A pointer to a memory area used to hold a parameter block. The format of this parameter block is device-specific and TAPI passes its contents to or from the service provider.

dwSize

The size in bytes of the parameter block area.

Return Values

Returns a positive request identifier if the function is completed asynchronously or a negative number if an error occurs. The dwParam2 parameter of the corresponding LINE_REPLY message is zero if the function succeeds or it is a negative number if an error occurs.

Possible return values follow:

- LINEERR_INVALFEATURE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPOINTER
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED.

Error Codes

The following new error can be returned by Cisco TSP for Do-Not-Disturb feature:

LINERR_ALREADY_IN_REQUESTED_STATE 0xC0000009

lineDial

The lineDial function dials the specified number on the specified call.

The application can use this function to enter a FAC or CMC. The FAC or CMC can be entered one digit at a time or multiple digits at a time. The application may also enter both the FAC and CMC if required in one lineDial() request as long as the FAC and CMC are separated by a “#” character. If sending both a FAC and

CMC in one lineDial() request, Cisco recommends that you terminate the lpszDestAddress with a "#" character to avoid waiting for the T.302 interdigit time-out.

You cannot use this function to enter a dial string along with a FAC and/or a CMC. You must enter the FAC and/or CMC in a separate lineDial request.

Function Details

```
LONG lineDial(  HCALL hCall,
  LPCSTR lpszDestAddress,
  DWORD dwCountryCode
);
```

Parameters

hCall

A handle to the call on which a number is to be dialed. Ensure the application is an owner of the call. The call state of hCall can be any state except idle and disconnected.

lpszDestAddress

The destination to be dialed by using the standard dial number format.

dwCountryCode

The country code of the destination. The implementation uses this code to select the call progress protocols for the destination address. If a value of 0 is specified, the default call progress protocol is used.

lineDrop

The lineDrop function drops or disconnects the specified call. The application can specify user-user information to be transmitted as part of the call disconnect.

Function Details

```
LONG lineDrop(  HCALL hCall,
  LPCSTR lpsUserUserInfo,
  DWORD dwSize
);
```

Parameters

hCall

A handle to the call to be dropped. Ensure the application is an owner of the call. The call state of hCall can be any state except an Idle state.

lpsUserUserInfo

A pointer to a string that contains user-user information to be sent to the remote party as part of the call disconnect. You can leave this pointer NULL if no user-user information is to be sent. User-user information is sent only if it is supported by the underlying network. The protocol discriminator field for the user-user information, if required, should appear as the first byte of the buffer that is pointed to by lpsUserUserInfo and must be accounted for in dwSize.

Note The Cisco Unified TSP does not support user-user information.

dwSize

The size in bytes of the user-user information in lpsUserUserInfo. If lpsUserUserInfo is NULL, no user-user information gets sent to the calling party, and dwSize is ignored.

lineForward

The lineForward function forwards calls that are destined for the specified address on the specified line, according to the specified forwarding instructions. When an originating address (dwAddressID) is forwarded, the switch deflects the specified incoming calls for that address to the other number. This function provides a combination of forward all feature. This API allows calls to be forwarded unconditionally to a forwarded destination. This function can also cancel forwarding that is currently in effect.

To indicate that the forward is set/reset, upon completion of lineForward, TAPI fires LINEADDRESSSTATE events that indicate the change in the line forward status.

Change forward destination with a call to lineForward without canceling the current forwarding set on that line.

Note lineForward implementation of Cisco Unified TSP allows user to set up only one type for forward as dwForwardMode = UNCOND. The lpLineForwardList data structure accepts LINEFORWARD entry with dwForwardMode = UNCOND.

Function Details

```
LONG lineForward(  HLINE hLine,
  DWORD bAllAddresses,
  DWORD dwAddressID,
  LPLINEFORWARDLIST const lpForwardList,
  DWORD dwNumRingsNoAnswer,
  LPHCALL lphConsultCall,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

hLine

A handle to the line device.

bAllAddresses

Specifies whether all originating addresses on the line or just the one that is specified gets forwarded. If TRUE, all addresses on the line get forwarded, and dwAddressID is ignored; if FALSE, only the address that is specified as dwAddressID gets forwarded.

dwAddressID

The address of the specified line whose incoming calls are to be forwarded. This parameter gets ignored if bAllAddresses is TRUE.

Note If bAllAddresses is FALSE, dwAddressID must equal 0.

lpForwardList

A pointer to a variably sized data structure that describes the specific forwarding instructions of type LINEFORWARDLIST.

Note To cancel the forwarding that currently is in effect, ensure lpForwardList Parameter is set to NULL.

dwNumRingsNoAnswer

The number of rings before a call is considered a no answer. If dwNumRingsNoAnswer is out of range, the actual value gets set to the nearest value in the allowable range.

Note This parameter is not used because this version of Cisco Unified TSP does not support call forward no answer.

lphConsultCall

A pointer to an HCALL location. In some telephony environments, this location is loaded with a handle to a consultation call that is used to consult the party to which the call is being forwarded, and the application becomes the initial sole owner of this call. This pointer must be valid even in environments where call forwarding does not require a consultation call. This handle is set to NULL if no consultation call is created.

Note This parameter is also ignored because a consult call is not created for setting up lineForward.

lpCallParams

A pointer to a structure of type LINECALLPARAMS. This pointer gets ignored unless lineForward requires the establishment of a call to the forwarding destination (and lphConsultCall is returned; in which case, lpCallParams is optional). If NULL, default call parameters get used. Otherwise, the specified call parameters get used for establishing hConsultCall.

Note This parameter must be NULL for this version of Cisco Unified TSP because we do not create a consult call.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALLINEHANDLE
- LINEERR_NOMEM
- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESS
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCOUNTRYCODE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_STRUCTURETOOSMALL
- LINEERR_INVALPARAM
- LINEERR_UNINITIALIZED

Note For lpForwardList[0].dwForwardMode other than UNCOND, lineForward returns LINEERR_OPERATIONUNAVAIL. For lpForwardList.dwNumEntries more than 1, lineForward returns LINEERR_INVALPARAM

lineGenerateDigits

The lineGenerateDigits function initiates the generation of the specified digits on the specified call as out-of-band tones by using the specified signaling mode.

Note The Cisco Unified TSP supports neither invoking this function with a NULL value for lpszDigits to abort a digit generation that is currently in progress nor invoking lineGenerateDigits while digit generation is in progress. Cisco Unified IP Phones pass DTMF digits out of band. This means that the tone is not injected into the audio stream (in-band) but is sent as a message in the control stream. The phone on the far end then injects the tone into the audio stream to present it to the user. CTI port devices do not inject DTMF tones. Also, be aware that some gateways will not inject DTMF tones into the audio stream on the way out of the LAN.

Function Details

```
LONG lineGenerateDigits(  HCALL hCall,
  DWORD dwDigitMode,
  LPCSTR lpszDigits,
  DWORD dwDuration
);
```

Parameters

hCall

A handle to the call. The application must be an owner of the call. Call state of hCall can be any state.

dwDigitMode

The format to be used for signaling these digits. The dwDigitMode can have only a single flag set. This parameter uses the following LINEDIGITMODE_constant:

- LINEDIGITMODE_DTMF -Uses DTMF tones for digit signaling. Valid digits for DTMF mode include ‘0’ -‘9’, ‘*’, ‘#’.

  lpszDigits

  Valid characters for DTMF mode in the Cisco Unified TSP include ‘0’ through ‘9’, ‘*’, and ‘#’.

  dwDuration

  Duration in milliseconds during which the tone should be sustained.

Note Cisco Unified TSP does not support dwDuration.

lineGenerateTone

The lineGenerateTone function generates the specified tone over the specified call.

Note The Cisco Unified TSP supports neither invoking this function with a 0 value for dwToneMode to abort a tone generation that is currently in progress nor invoking lineGenerateTone while tone generation is in progress. Cisco Unified IP Phones pass tones out of band. This means that the tone is not injected into the audio stream (in-band) but is sent as a message in the control stream. The phone on the far end then injects the tone into the audio stream to present it to the user. Also, be aware that some gateways will not inject tones into the audio stream on the way out of the LAN.

Function Details

```
LONG lineGenerateTone(  HCALL hCall,
  DWORD dwToneMode,
  DWORD dwDuration,
  DWORD dwNumTones,
  LPLINEGENERATETONE const lpTones
);
```

Parameters

hCall

A handle to the call on which a tone is to be generated. The application must be an owner of the call. The call state of hCall can be any state.

dwToneMode

Defines the tone to be generated. Tones can be either standard or custom tones. A custom tone comprises a set of arbitrary frequencies. A small number of standard tones are predefined. The duration of the tone gets specified with dwDuration for both standard and custom tones. The dwToneMode parameter can have only one bit set. If no bits are set (the value 0 is passed), tone generation gets canceled.

This parameter uses the following LINETONEMODE_constant:

- LINETONEMODE_BEEP -The tone is a beep, as used to announce the beginning of a recording. The service provider defines the exact beep tone.

dwDuration

Duration in milliseconds during which the tone should be sustained.

Note Cisco Unified TSP does not support dwDuration.

dwNumTones

The number of entries in the lpTones array. This parameter is ignored if dwToneMode ≠ CUSTOM.

lpTones

A pointer to a LINEGENERATETONE array that specifies the components of the tone. This parameter gets ignored for non-custom tones. If lpTones is a multifrequency tone, the various tones play simultaneously.

lineGetAddressCaps

The lineGetAddressCaps function queries the specified address on the specified line device to determine its telephony capabilities.

Function Details

```
LONG lineGetAddressCaps(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAddressID,
  DWORD dwAPIVersion,
  DWORD dwExtVersion,
  LPLINEADDRESSCAPS lpAddressCaps
);
```

Parameters

hLineApp

The handle by which the application is registered with TAPI.

dwDeviceID

The line device that contains the address to be queried. Only one address gets supported per line, so dwAddressID must be zero.

dwAddressID

The address on the given line device whose capabilities are to be queried.

dwAPIVersion

The version number, obtained by lineNegotiateAPIVersion, of the API that is to be used. The high-order word contains the major version number; the low-order word contains the minor version number.

dwExtVersion

The version number of the extensions to be used. This number can be left zero if no device-specific extensions are to be used. Otherwise, the high-order word contains the major version number and the low-order word contains the minor version number.

lpAddressCaps

A pointer to a variably sized structure of type LINEADDRESSCAPS. Upon successful completion of the request, this structure gets filled with address capabilities information. Prior to calling lineGetAddressCaps, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lineGetAddressID

The lineGetAddressID function returns the address identifier that is associated with an address in a different format on the specified line.

Function Details

```
LONG lineGetAddressID(  HLINE hLine,
  LPDWORD lpdwAddressID,
  DWORD dwAddressMode,
  LPCSTR lpsAddress,
  DWORD dwSize
);
```

Parameters

hLine

A handle to the open line device.

lpdwAddressID

A pointer to a DWORD-sized memory location that returns the address identifier.

dwAddressMode

The address mode of the address that is contained in lpsAddress. The dwAddressMode parameter can have only a single flag set. This parameter uses the following LINEADDRESSMODE_constant:

- LINEADDRESSMODE_DIALABLEADDR -The address is specified by its dialable address. The lpsAddress parameter represents the dialable address or canonical address format.

lpsAddress

A pointer to a data structure that holds the address that is assigned to the specified line device. dwAddressMode determines the format of the address. Because the only valid value equals

LINEADDRESSMODE_DIALABLEADDR, lpsAddress uses the common dialable number format and is NULL-terminated.

dwSize

The size of the address that is contained in lpsAddress.

lineGetAddressStatus

The lineGetAddressStatus function allows an application to query the specified address for its current status.

Function Details

```
LONG lineGetAddressStatus(  HLINE hLine,
  DWORD dwAddressID,
  LPLINEADDRESSSTATUS lpAddressStatus
);
```

Parameters

hLine

A handle to the open line device.

dwAddressID

An address on the given open line device. This parameter specifies the address to be queried.

lpAddressStatus

A pointer to a variably sized data structure of type LINEADDRESSSTATUS. Prior to calling lineGetAddressStatus, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lineGetCallInfo

The lineGetCallInfo function enables an application to obtain fixed information about the specified call.

Function Details

```
LONG lineGetCallInfo(  HCALL hCall,
  LPLINECALLINFO lpCallInfo
);
```

Parameters

hCall

A handle to the call to be queried. The call state of hCall can be any state.

lpCallInfo

A pointer to a variably sized data structure of type LINECALLINFO. Upon successful completion of the request, call-related information fills this structure. Prior to calling lineGetCallInfo, the application

should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lineGetCallStatus

The lineGetCallStatus function returns the current status of the specified call.

Function Details

```
LONG lineGetCallStatus(  HCALL hCall,
  LPLINECALLSTATUS lpCallStatus
);
```

Parameters

hCall

A handle to the call to be queried. The call state of hCall can be any state.

lpCallStatus

A pointer to a variably sized data structure of type LINECALLSTATUS. Upon successful completion of the request, call status information fills this structure. Prior to calling lineGetCallStatus, the application should set the dwTotalSize member of this structure to indicate the amount of memory available to TAPI for returning information.

lineGetConfRelatedCalls

The lineGetConfRelatedCalls function returns a list of call handles that are part of the same conference call as the specified call. The specified call represents either a conference call or a participant call in a conference call. New handles get generated for those calls for which the application does not already have handles, and the application receives monitor privilege to those calls.

Function Details

```
LONG WINAPI lineGetConfRelatedCalls(  HCALL hCall,
  LPLINECALLLIST lpCallList
);
```

Parameters

hCall

A handle to a call. This represents either a conference call or a participant call in a conference call. For a conference parent call, the call state of hCall can be any state. For a conference participant call, it must be in the conferenced state.

lpCallList

A pointer to a variably sized data structure of type LINECALLLIST. Upon successful completion of the request, call handles to all calls in the conference call return in this structure. The first call in the list represents the conference call, the other calls represent the participant calls. The application receives monitor privilege to those calls for which it does not already have handles; the privileges to calls in the

list for which the application already has handles remains unchanged. Prior to calling lineGetConfRelatedCalls, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

Return Values

Returns zero if request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_NOCONFERENCE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_STRUCTURETOOSMALL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED

lineGetDevCaps

The lineGetDevCaps function queries a specified line device to determine its telephony capabilities. The returned information applies for all addresses on the line device.

Function Details

```
LONG lineGetDevCaps(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAPIVersion,
  DWORD dwExtVersion,
  LPLINEDEVCAPS lpLineDevCaps
);
```

Parameters

hLineApp

The handle by which the application is registered with TAPI.

dwDeviceID

The line device to be queried.

dwAPIVersion

The version number, obtained by lineNegotiateAPIVersion, of the API to be used. The high-order word contains the major version number; the low-order word contains the minor version number.

dwExtVersion

The version number, obtained by lineNegotiateExtVersion, of the extensions to be used. It can be zero if no device-specific extensions are to be used. Otherwise, the high-order word contains the major version number; the low-order word contains the minor version number.

lpLineDevCaps

A pointer to a variably sized structure of type LINEDEVCAPS. Upon successful completion of the request, this structure gets filled with line device capabilities information. Prior to calling lineGetDevCaps, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lineGetID

The lineGetID function returns a device identifier for the specified device class that is associated with the selected line, address, or call.

Function Details

```
LONG lineGetID(  HLINE hLine,
  DWORD dwAddressID,
  HCALL hCall,
  DWORD dwSelect,
  LPVARSTRING lpDeviceID,
  LPCSTR lpszDeviceClass
);
```

Parameters

hLine

A handle to an open line device.

dwAddressID

An address on the given open line device.

hCall

A handle to a call.

dwSelect

Specifies whether the requested device identifier is associated with the line, address or a single call. The dwSelect parameter can only have a single flag set. This parameter uses the following LINECALLSELECT_constants:

- LINECALLSELECT_LINE Selects the specified line device. The hLine parameter must be a valid line handle; hCall and dwAddressID are ignored.
- LINECALLSELECT_ADDRESS Selects the specified address on the line. Both hLine and dwAddressID must be valid; hCall is ignored.
- LINECALLSELECT_CALL Selects the specified call. hCall must be valid; hLine and dwAddressID are both ignored.

lpDeviceID

A pointer to a memory location of type VARSTRING, where the device identifier is returned. Upon successful completion of the request, the device identifier fills this location. The format of the returned information depends on the method that the device class API uses for naming devices. Before calling

lineGetID, the application must set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lpszDeviceClass

A pointer to a NULL-terminated ASCII string that specifies the device class of the device whose identifier is requested. Device classes include wave/in, wave/out and tapi/line.

Valid device class strings are those that are used in the SYSTEM.INI section to identify device classes.

lineGetLineDevStatus

The lineGetLineDevStatus function enables an application to query the specified open line device for its current status.

Function Details

```
LONG lineGetLineDevStatus(  HLINE hLine,
  LPLINEDEVSTATUS lpLineDevStatus
);
```

Parameters

hLine

A handle to the open line device to be queried.

lpLineDevStatus

A pointer to a variably sized data structure of type LINEDEVSTATUS. Upon successful completion of the request, the device status of the line fills this structure. Prior to calling lineGetLineDevStatus, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

lineGetMessage

The lineGetMessage function returns the next TAPI message that is queued for delivery to an application that is using the Event Handle notification mechanism (see lineInitializeEx, on page 172 for more information).

Function Details

```
LONG WINAPI lineGetMessage(  HLINEAPP hLineApp,
  LPLINEMESSAGE lpMessage,
  DWORD dwTimeout
);
```

Parameters

hLineApp

The handle that lineInitializeEx returns. Ensure that the application has set the LINEINITIALIZEEXOPTION_USEEVENT option in the dwOptions member of the LINEINITIALIZEEXPARAMS structure.

lpMessage

A pointer to a LINEMESSAGE structure. Upon successful return from this function, the structure contains the next message that had been queued for delivery to the application.

dwTimeout

The time-out interval, in milliseconds. The function returns if the interval elapses, even if no message can be returned. If dwTimeout is zero, the function checks for a queued message and returns immediately. If dwTimeout is INFINITE, the function time-out interval never elapses.

Return Values

Returns zero if the request succeeds or returns a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALAPPHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPOINTER
- LINEERR_NOMEM

lineGetNewCalls

The lineGetNewCalls function returns call handles to calls on a specified line or address for which the application currently does not have handles. The application receives monitor privilege for these calls.

An application can use lineGetNewCalls to obtain handles to calls for which it currently has no handles. The application can select the calls for which handles are to be returned by basing this selection on scope (calls on a specified line, or calls on a specified address). For example, an application can request call handles to all calls on a given address for which it currently has no handle.

Function Details

```
LONG WINAPI lineGetNewCalls(  HLINE hLine,
  DWORD dwAddressID,
  DWORD dwSelect,
  LPLINECALLLIST lpCallList
);
```

Parameters

hLine

A handle to an open line device.

dwAddressID

An address on the given open line device. An address identifier permanently associates with an address; the identifier remains constant across operating system upgrades.

dwSelect

The selection of calls that are requested. This parameter uses one and only one of the LINECALLSELECT_Constants.

lpCallList

A pointer to a variably sized data structure of type LINECALLLIST. Upon successful completion of the request, call handles to all selected calls get returned in this structure. Prior to calling lineGetNewCalls, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCALLSELECT
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALLINEHANDLE
- LINEERR_STRUCTURETOOSMALL
- LINEERR_INVALPOINTER
- LINEERR_UNINITIALIZED
- LINEERR_NOMEM

lineGetNumRings

The lineGetNumRings function determines the number of rings that an incoming call on the given address should ring before the call is answered.

Function Details

```
LONG WINAPI lineGetNumRings(  HLINE hLine,
  DWORD dwAddressID,
  LPDWORD lpdwNumRings
);
```

Parameters

hLine

A handle to the open line device.

dwAddressID

An address on the line device. An address identifier permanently associates with an address; the identifier remains constant across operating system upgrades.

lpdwNumRings

The number of rings that is the minimum of all current lineSetNumRings requests.

Return Values

Returns zero if request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONFAILED
- LINEERR_INVALLINEHANDLE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_UNINITIALIZED
- LINEERR_NOMEM

lineGetProviderList

The lineGetProviderList function returns a list of service providers that are currently installed in the telephony system.

Function Details

```
LONG WINAPI lineGetProviderList(  DWORD dwAPIVersion,
  LPLINEPROVIDERLIST lpProviderList
);
```

Parameters

dwAPIVersion

The highest version of TAPI that the application supports (not necessarily the value that lineNegotiateAPIVersion negotiates on some particular line device).

lpProviderList

A pointer to a memory location where TAPI can return a LINEPROVIDERLIST structure. Prior to calling lineGetProviderList, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

Return Values

Returns zero if request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_NOMEM
- LINEERR_INIFILECORRUPT
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPOINTER
- LINEERR_STRUCTURETOOSMALL

lineGetRequest

The lineGetRequest function retrieves the next by-proxy request for the specified request mode.

Function Details

```
LONG WINAPI lineGetRequest(  HLINEAPP hLineApp,
  DWORD dwRequestMode,
  LPVOID lpRequestBuffer
);
```

Parameters

hLineApp

The application's usage handle for the line portion of TAPI.

dwRequestMode

The type of request that is to be obtained. dwRequestMode can have only one bit set. This parameter uses one and only one of the LINEREQUESTMODE_Constants.

lpRequestBuffer

A pointer to a memory buffer where the parameters of the request are to be placed. The size of the buffer and the interpretation of the information that is placed in the buffer depends on the request mode. The application-allocated buffer provides sufficient size to hold the request. If dwRequestMode is LINEREQUESTMODE_MAKECALL, interpret the content of the request buffer by using the LINEREQMAKECALL structure. If dwRequestMode is LINEREQUESTMODE_MEDIACALL, interpret the content of the request buffer by using the LINEREQMEDIACALL structure.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALAPPHANDLE
- LINEERR_NOTREGISTERED
- LINEERR_INVALPOINTER
- LINEERR_OPERATIONFAILED
- LINEERR_INVALREQUESTMODE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED
- LINEERR_NOREQUEST

lineGetStatusMessages

The lineGetStatusMessages function enables an application to query the notification messages that the application receives for events related to status changes for the specified line or any of its addresses.

Function Details

```
LONG WINAPI lineGetStatusMessages(  HLINE hLine,
  LPDWORD lpdwLineStates,
  LPDWORD lpdwAddressStates
);
```

Parameters

hLine

Handle to the line device.

lpdwLineStates

A bit array that identifies the line device status changes for which a message is to be sent to the application. If a flag is TRUE, that message is enabled; if FALSE, it is disabled. This parameter uses one or more LINEDEVSTATE_Constants.

lpdwAddressStates

A bit array that identifies for which address status changes a message is to be sent to the application. If a flag is TRUE, that message is enabled; if FALSE, disabled. This parameter uses one or more LINEADDRESSSTATE_Constants.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPOINTER
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED

lineGetTranslateCaps

The lineGetTranslateCaps function returns address translation capabilities.

Function Details

```
LONG WINAPI lineGetTranslateCaps(  HLINEAPP hLineApp,
  DWORD dwAPIVersion,
```

```
  LPLINETRANSLATECAPS lpTranslateCaps
);
```

Parameters

hLineApp

The application handle that lineInitializeEx returns. If an application has not yet called the lineInitializeEx function, it can set the hLineApp parameter to NULL.

dwAPIVersion

The highest version of TAPI that the application supports (not necessarily the value that lineNegotiateAPIVersion negotiates on some particular line device).

lpTranslateCaps

A pointer to a location to which a LINETRANSLATECAPS structure is loaded. Prior to calling lineGetTranslateCaps, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_NOMEM
- LINEERR_INIFILECORRUPT
- LINEERR_OPERATIONFAILED
- LINEERR_INVALAPPHANDLE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_STRUCTURETOOSMALL
- LINEERR_NODRIVER.

lineHandoff

The lineHandoff function gives ownership of the specified call to another application. Specify the application either directly by its file name or indirectly as the highest priority application that handles calls of the specified media mode.

Function Details

```
LONG WINAPI lineHandoff(  HCALL hCall,
  LPCSTR lpszFileName,
  DWORD dwMediaMode
);
```

Parameters

hCall

A handle to the call to be handed off. The application must be an owner of the call. The call state of hCall can be any state.

lpszFileName

A pointer to a null-terminated string. If this pointer parameter is non-NULL, it contains the file name of the application that is the target of the handoff. If NULL, the handoff target represents the highest priority application that has opened the line for owner privilege for the specified media mode. A valid file name does not include the path of the file.

dwMediaMode

The media mode that is used to identify the target for the indirect handoff. The dwMediaMode parameter indirectly identifies the target application that is to receive ownership of the call. This parameter gets ignored if lpszFileName is not NULL. This parameter uses one and only one of the LINEMEDIAMODE_Constants.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALMEDIAMODE
- LINEERR_TARGETNOTFOUND
- LINEERR_INVALPOINTER
- LINEERR_TARGETSELF
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED
- LINEERR_NOTOWNER

lineHold

The lineHold function places the specified call on hold.

Function Details

```
LONG lineHold(  HCALL hCall
);
```

Parameter

hCall

A handle to the call that is to be placed on hold. Ensure that the application is an owner of the call and the call state of hCall is connected.

lineInitialize

Although the lineInitialize function is obsolete, tapi.dll and tapi32.dll continue to export it for backward compatibility with applications that are using API versions 1.3 and 1.4.

Function Details

```
LONG WINAPI lineInitialize(  LPHLINEAPP lphLineApp,
  HINSTANCE hInstance,
  LINECALLBACK lpfnCallback,
  LPCSTR lpszAppName,
  LPDWORD lpdwNumDevs
);
```

Parameters

lphLineApp

A pointer to a location that is filled with the application's usage handle for TAPI.

hInstance

The instance handle of the client application or DLL.

lpfnCallback

The address of a callback function that is invoked to determine status and events on the line device, addresses, or calls. For more information, see lineCallbackFunc.

lpszAppName

A pointer to a null-terminated text string that contains only displayable characters. If this parameter is not NULL, it contains an application-supplied name for the application. The LINECALLINFO structure provides this name to indicate, in a user-friendly way, which application originated, originally accepted, or answered the call. This information can prove useful for call logging purposes. If lpszAppName is NULL, the application's file name gets used instead.

lpdwNumDevs

A pointer to a DWORD-sized location. Upon successful completion of this request, this location gets filled with the number of line devices that is available to the application.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALAPPNAME
- LINEERR_OPERATIONFAILED
- LINEERR_INIFILECORRUPT
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER

- LINEERR_REINIT
- LINEERR_NODRIVER
- LINEERR_NODEVICE
- LINEERR_NOMEM
- LINEERR_NOMULTIPLEINSTANCE.

lineInitializeEx

The lineInitializeEx function initializes the use of TAPI by the application for the subsequent use of the line abstraction. It registers the specified notification mechanism of the application and returns the number of line devices that are available. A line device represents any device that provides an implementation for the line-prefixed functions in the telephony API.

Function Details

```
LONG lineInitializeEx(  LPHLINEAPP lphLineApp,
  HINSTANCE hInstance,
  LINECALLBACK lpfnCallback,
  LPCSTR lpszFriendlyAppName,
  LPDWORD lpdwNumDevs,
  LPDWORD lpdwAPIVersion,
  LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams
);
```

Parameters

lphLineApp

A pointer to a location that is filled with the TAPI usage handle for the application.

hInstance

The instance handle of the client application or DLL. The application or DLL can pass NULL for this parameter, in which case TAPI uses the module handle of the root executable of the process (for purposes of identifying call handoff targets and media mode priorities).

lpfnCallback

The address of a callback function that is invoked to determine status and events on the line device, addresses, or calls, when the application is using the “hidden window” method of event notification. This parameter gets ignored and should be set to NULL when the application chooses to use the “event handle” or “completion port” event notification mechanisms.

lpszFriendlyAppName

A pointer to a NULL-terminated ASCII string that contains only standard ASCII characters. If this parameter is not NULL, it contains an application-supplied name for the application. The LINECALLINFO structure provides this name to indicate, in a user-friendly way, which application originated, originally accepted, or answered the call. This information can prove useful for call-logging purposes. If lpszFriendlyAppName is NULL, the module filename of the application gets used instead (as returned by the Windows API GetModuleFileName).

lpdwNumDevs

A pointer to a DWORD-sized location. Upon successful completion of this request, this location gets filled with the number of line devices that are available to the application.

lpdwAPIVersion

A pointer to a DWORD-sized location. The application must initialize this DWORD, before calling this function, to the highest API version that it is designed to support (for example, the same value that it would pass into dwAPIHighVersion parameter of lineNegotiateAPIVersion). Make sure that artificially high values are not used; ensure that the value is set to 0x00020000. TAPI translates any newer messages or structures into values or formats that the application supports. Upon successful completion of this request, this location is filled with the highest API version that TAPI supports, which allows the application to adapt to being installed on a system with an older TAPI version.

lpLineInitializeExParams

A pointer to a structure of type LINEINITIALIZEEXPARAMS that contains additional parameters that are used to establish the association between the application and TAPI (specifically, the selected event notification mechanism of the application and associated parameters).

lineMakeCall

The lineMakeCall function places a call on the specified line to the specified destination address. Optionally, you can specify call parameters if anything but default call setup parameters are requested.

Function Details

```
LONG lineMakeCall(HLINE hLine,
LPHCALL lphCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode,
LPLINECALLPARAMS const lpCallParams
);
typedef struct LineParams {
DWORD FeaturePriority;
```

Parameters

hLine

A handle to the open line device on which a call is to be originated.

lphCall

A pointer to an HCALL handle. The handle is only valid after the application receives LINE_REPLY message that indicates that the lineMakeCall function successfully completed. Use this handle to identify the call when you invoke other telephony operations on the call. The application initially acts as the sole owner of this call. This handle registers as void if the reply message returns an error (synchronously or asynchronously).

lpszDestAddress

A pointer to the destination address. This parameter follows the standard dialable number format. This pointer can be NULL for non-dialed addresses or when all dialing is performed by using lineDial. In the latter case, lineMakeCall allocates an available call appearance that would typically remain in the dial tone state until dialing begins.

dwCountryCode

The country code of the called party. If a value of 0 is specified, the implementation uses a default.

lpCallParams

The dwNoAnswerTimeout attribute of the lpCallParams field is checked and if specified as non-zero, automatically disconnects a call if not answered after the specified time. For more information, see LINECALLPARAMS, on page 336.

lineMonitorDigits

The lineMonitorDigits function enables and disables the unbuffered detection of digits that are received on the call. Each time that a digit of the specified digit mode is detected, a message gets sent to the application to indicate which digit has been detected.

Function Details

```
LONG lineMonitorDigits(  HCALL hCall,
  DWORD dwDigitModes
);
```

Parameters

hCall

A handle to the call on which digits are to be detected. The call state of hCall can be any state except idle or disconnected.

dwDigitModes

The digit mode or modes that are to be monitored. If dwDigitModes is zero, the system cancels digit monitoring. This parameter which can have multiple flags set, uses the following LINEDIGITMODE_constant:

LINEDIGITMODE_DTMF -Detect digits as DTMF tones. Valid digits for DTMF include ‘0’ through ‘9’, ‘*’, and ‘#’.

lineMonitorTones

The lineMonitorTones function enables and disables the detection of inband tones on the call. Each time that a specified tone is detected, a message gets sent to the application.

Function Details

```
LONG lineMonitorTones(  HCALL hCall,
  LPLINEMONITORTONE const lpToneList,
  DWORD dwNumEntries
);
```

Parameters

hCall

A handle to the call on which tones are to be detected. The call state of hCall can be any state except idle.

lpToneList

A list of tones to be monitored, of type LINEMONITORTONE. Each tone in this list has an application-defined tag field that is used to identify individual tones in the list to report a tone detection. Calling this operation with either NULL for lpToneList or with another tone list cancels or changes tone monitoring in progress.

dwNumEntries

The number of entries in lpToneList. This parameter gets ignored if lpToneList is NULL.

lineNegotiateAPIVersion

The lineNegotiateAPIVersion function allows an application to negotiate an API version to use. The Cisco Unified TSP supports TAPI 2.0 and 2.1.

Function Details

```
LONG lineNegotiateAPIVersion(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAPILowVersion,
  DWORD dwAPIHighVersion,
  LPDWORD lpdwAPIVersion,
  LPLINEEXTENSIONID lpExtensionID
);
```

Parameters

hLineApp

The handle by which the application is registered with TAPI.

dwDeviceID

The line device to be queried.

dwAPILowVersion

The least recent API version with which the application is compliant. The high-order word specifies the major version number; the low-order word specifies the minor version number.

dwAPIHighVersion

The most recent API version with which the application is compliant. The high-order word specifies the major version number; the low-order word specifies the minor version number.

lpdwAPIVersion

A pointer to a DWORD-sized location that contains the API version number that was negotiated. If negotiation succeeds, this number falls in the range between dwAPILowVersion and dwAPIHighVersion.

lpExtensionID

A pointer to a structure of type LINEEXTENSIONID. If the service provider for the specified dwDeviceID supports provider-specific extensions, upon a successful negotiation, this structure gets filled with the extension identifier of these extensions. This structure contains all zeros if the line provides no extensions. An application can ignore the returned parameter if it does not use extensions.

The Cisco Unified TSP extensionID specifies 0x8EBD6A50, 0x138011d2, 0x905B0060, 0xB03DD275.

lineNegotiateExtVersion

The lineNegotiateExtVersion function allows an application to negotiate an extension version to use with the specified line device. Do not call this operation if the application does not support extensions.

Function Details

```
LONG lineNegotiateExtVersion(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAPIVersion,
  DWORD dwExtLowVersion,
  DWORD dwExtHighVersion,
  LPDWORD lpdwExtVersion
);
```

Parameters

hLineApp

The handle by which the application is registered with TAPI.

dwDeviceID

The line device to be queried.

dwAPIVersion

The API version number that was negotiated for the specified line device by using lineNegotiateAPIVersion.

dwExtLowVersion

The least recent extension version of the extension identifier that lineNegotiateAPIVersion returns and with which the application is compliant. The high-order word specifies the major version number; the low-order word specifies the minor version number.

dwExtHighVersion

The most recent extension version of the extension identifier that lineNegotiateAPIVersion returns and with which the application is compliant. The high-order word specifies the major version number; the low-order word specifies the minor version number.

lpdwExtVersion

A pointer to a DWORD-sized location that contains the extension version number that was negotiated. If negotiation succeeds, this number falls between dwExtLowVersion and dwExtHighVersion.

lineOpen

The lineOpen function opens the line device that its device identifier specifies and returns a line handle for the corresponding opened line device. Subsequent operations on the line device use this line handle.

Function Details

```
LONG lineOpen(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  LPHLINE lphLine,
  DWORD dwAPIVersion,
  DWORD dwExtVersion,
  DWORD dwCallbackInstance,
  DWORD dwPrivileges,
  DWORD dwMediaModes,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

hLineApp

The handle by which the application is registered with TAPI.

dwDeviceID

Identifies the line device to be opened. It can either be a valid device identifier or the value LINEMAPPER

Note The Cisco Unified TSP does not support LINEMAPPER at this time.

lphLine

A pointer to an HLINE handle that is then loaded with the handle that represents the opened line device. Use this handle to identify the device when you are invoking other functions on the open line device.

dwAPIVersion

The API version number under which the application and Telephony API operate. Obtain this number with lineNegotiateAPIVersion.

dwExtVersion

The extension version number under which the application and the service provider operate. This number remains zero if the application does not use any extensions. Obtain this number with lineNegotiateExtVersion.

dwCallbackInstance

User-instance data that is passed back to the application with each message that is associated with this line or with addresses or calls on this line. The Telephony API does not interpret this parameter.

dwPrivileges

The privilege that the application wants for the calls for which it is notified. This parameter can be a combination of the LINECALLPRIVILEGE_ constants. For applications that are using TAPI version 2.0 or later, values for this parameter can also be combined with the LINEOPENOPTION_constants:

- LINECALLPRIVILEGE_NONE -The application can make only outgoing calls.
- LINECALLPRIVILEGE_MONITOR -The application can monitor only incoming and outgoing calls.
- LINECALLPRIVILEGE_OWNER -The application can own only incoming calls of the types that are specified in dwMediaModes.
- LINECALLPRIVILEGE_MONITOR + LINECALLPRIVILEGE_OWNER -The application can own only incoming calls of the types that are specified in dwMediaModes, but if the application does not represent an owner of a call, it acts as a monitor.
- Other flag combinations return the LINEERR_INVALPRIVSELECT error.

dwMediaModes

The media mode or modes of interest to the application. Use this parameter to register the application as a potential target for incoming call and call handoff for the specified media mode. This parameter proves meaningful only if the bit LINECALLPRIVILEGE_OWNER in dwPrivileges is set (and ignored if it is not).

This parameter uses the following LINEMEDIAMODE_constant:

- LINEMEDIAMODE_INTERACTIVEVOICE -The application can handle calls of the interactive voice media type; that is, it manages voice calls with the user on this end of the call. Use this parameter for third-party call control of physical phones and CTI port and CTI route point devices that other applications opened.
- LINEMEDIAMODE_AUTOMATEDVOICE -Voice energy exists on the call. An automated application locally handles the voice. This represents first-party call control and is used with CTI port and CTI route point devices.

lpCallParams

The dwNoAnswerTimeout attribute of the lpCallParams field is checked and if it is non-zero, automatically disconnects a call if it is not answered after the specified time.

linePark

The linePark function parks the specified call according to the specified park mode.

Function Details

```
LONG WINAPI linePark(HCALL hCall,
DWORD dwParkMode,
LPCSTR lpszDirAddress,
LPVARSTRING lpNonDirAddress
);
```

Parameters

hCall

Handle to the call to be parked. The application must act as an owner of the call. The call state of hcall must be connected.

dwParkMode

Park mode with which the call is parked. This parameter can have only a single flag set and uses one of the LINEPARKMODE_Constants.

Note Ensure that LINEPARKMODE_Constants is set to LINEPARKMODE_NONDIRECTED.

lpszDirAddress

Pointer to a null-terminated string that indicates the address where the call is to be parked when directed park is used. The address specifies in dialable number format. This parameter gets ignored for nondirected park.

Note This parameter gets ignored.

lpNonDirAddress

Pointer to a structure of type VARSTRING. For nondirected park, the address where the call is parked gets returned in this structure. This parameter gets ignored for directed park. Within the VARSTRING structure, ensure that dwStringFormat is set to STRINGFORMAT_ASCII (an ASCII string buffer that contains a null-terminated string), and the terminating NULL must be accounted for in the dwStringSize. Before calling linePark, the application must set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

linePrepareAddToConference

The linePrepareAddToConference function prepares an existing conference call for the addition of another party.

If LINEERR_INVALLINESTATE is returned, that means that the line is currently not in a state in which this operation can be performed. The dwLineFeatures member includes a list of currently valid operations (of the type LINEFEATURE) in the LINEDEVSTATUS structure. (Calling lineGetLineDevStatus updates the information in LINEDEVSTATUS.)

Obtain a conference call handle with lineSetupConference or with lineCompleteTransfer that is resolved as a three-way conference call. The linePrepareAddToConference function typically places the existing conference call in the onHoldPendingConference state and creates a consultation call that can be added later to the existing conference call with lineAddToConference.

You can cancel the consultation call by using lineDrop. You may also be able to swap an application between the consultation call and the held conference call with lineSwapHold.

Function Details

```
LONG WINAPI linePrepareAddToConference(  HCALL hConfCall,
  LPHCALL lphConsultCall,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

hConfCall

A handle to a conference call. The application must act as an owner of this call. Ensure that the call state of hConfCall is connected.

lphConsultCall

A pointer to an HCALL handle. This location then gets loaded with a handle that identifies the consultation call to be added. Initially, the application serves as the sole owner of this call.

lpCallParams

A pointer to call parameters that gets used when the consultation call is established. You can set this parameter to NULL if no special call setup parameters are desired.

Return Values

Returns a positive request identifier if the function completes asynchronously, or a negative number if an error occurs. The dwParam2 parameter of the corresponding LINE_REPLY message specifies zero if the function succeeds, or it is a negative number if an error occurs.

Possible return values follow:

- LINEERR_BEARERMODEUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_CALLUNAVAIL
- LINEERR_INVALRATE
- LINEERR_CONFERENCEFULL
- LINEERR_NOMEM
- LINEERR_INUSE
- LINEERR_NOTOWNER
- LINEERR_INVALADDRESSMODE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALBEARERMODE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCALLPARAMS
- LINEERR_RATEUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCONFCALLHANDLE
- LINEERR_STRUCTURETOOSMALL
- LINEERR_INVALLINESTATE

- LINEERR_USERUSERINFOTOOBIG
- LINEERR_INVALMEDIAMODE
- LINEERR_UNINITIALIZED

lineRedirect

The lineRedirect function redirects the specified offered or accepted call to the specified destination address.

Note If the application tries to redirect a call to an address that requires a FAC, CMC, or both, the lineRedirect function returns an error. If a FAC is required, the TSP returns the message LINEERR_FACREQUIRED. If a CMC is required, the TSP returns the message LINEERR_CMCREQUIRED. If both a FAC and a CMC are required, the TSP returns the message LINEERR_FACANDCMCREQUIRED. An application that wants to redirect a call to an address that requires a FAC, CMC, or both, should use the lineDevSpecific RedirectFACCMC function.

Function Details

```
LONG lineRedirect(  HCALL hCall,
  LPCSTR lpszDestAddress,
  DWORD dwCountryCode
);
```

Parameters

hCall

A handle to the call to be redirected. The application must act as an owner of the call. The call state of hCall must be offering, accepted, or connected.

Note The Cisco Unified TSP supports redirecting of calls in the connected call state.

lpszDestAddress

A pointer to the destination address. This follows the standard dialable number format.

dwCountryCode

The country code of the party to which the call is redirected. If a value of 0 is specified, the implementation uses a default.

lineRegisterRequestRecipient

The lineRegisterRequestRecipient function registers the invoking application as a recipient of requests for the specified request mode.

Function Details

```
LONG WINAPI lineRegisterRequestRecipient(  HLINEAPP hLineApp,
  DWORD dwRegistrationInstance,
  DWORD dwRequestMode,
  DWORD bEnable
);
```

Parameters

hLineApp

The application's usage handle for the line portion of TAPI.

dwRegistrationInstance

An application-specific DWORD that is passed back as a parameter of the LINE_REQUEST message. This message notifies the application that a request is pending. This parameter gets ignored if bEnable is set to zero. TAPI examines this parameter only for registration, not for deregistration. The dwRegistrationInstance value that is used while deregistering need not match the dwRegistrationInstance that is used while registering for a request mode.

dwRequestMode

The type or types of request for which the application registers. This parameter uses one or more LINEREQUESTMODE_Constants.

bEnable

If TRUE, the application registers the specified request modes; if FALSE, the application deregisters for the specified request modes.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALAPPHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALREQUESTMODE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED

lineRemoveFromConference

The lineRemoveFromConference function removes a specified call from the conference call to which it currently belongs. The remaining calls in the conference call are unaffected.

Function Details

```
LONG WINAPI lineRemoveFromConference(  HCALL hCall
);
```

Parameters

hCall

Handle to the call that is to be removed from the conference. The application must be an owner of this call. The call state of hCall must be conference.

Return Values

Returns a positive request identifier if the function is completed asynchronously, or a negative number if an error occurs. The dwParam2 parameter of the corresponding LINE_REPLY message is zero if the function succeeds or it is a negative number if an error occurs. The following table shows the return values for this function:

Value	Description
LINEERR_INVALCALLHANDLE	The handle to the call that is to be removed is invalid.
LINEERR_OPERATIONUNAVAIL	The operation is unavailable.
LINEERR_INVALCALLSTATE	The call state is something other than conferenced.
LINEERR_OPERATIONFAILED	The operation failed.
LINEERR_NOMEM	Not enough memory.
LINEERR_RESOURCEUNAVAIL	The resources are unavailable.
LINEERR_NOTOWNER	The application is not the owner of this call.
LINEERR_UNINITIALIZED	A parameter is uninitialized.

lineRemoveProvider

The lineRemoveProvider function removes an existing telephony service provider from the system.

Function Details

```
LONG WINAPI lineRemoveProvider(  DWORD dwPermanentProviderID,
  HWND hwndOwner
);
```

Parameters

dwPermanentProviderID

The permanent provider identifier of the service provider that is to be removed.

hwndOwner

A handle to a window to which any dialog boxes that need to be displayed as part of the removal process (for example, a confirmation dialog box by the service provider's TSPI_providerRemove function) would be attached. The parameter can be a NULL value to indicate that any window that is created during the function should have no owner window.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INIFILECORRUPT
- LINEERR_NOMEM
- LINEERR_INVALPARAM
- LINEERR_OPERATIONFAILED

lineSetAppPriority

The lineSetAppPriority function allows an application to set its priority in the handoff priority list for a particular media type or Assisted Telephony request mode or to remove itself from the priority list.

Function Details

```
LONG WINAPI lineSetAppPriority(  LPCSTR lpszAppFilename,
  DWORD dwMediaMode,
  LPLINEEXTENSIONID lpExtensionID,
  DWORD dwRequestMode,
  LPCSTR lpszExtensionName,
  DWORD dwPriority
);
```

Parameters

lpszAppFilename

A pointer to a string that contains the application executable module filename (without directory information). In TAPI version 2.0 or later, the parameter can specify a filename in either long or 8.3 filename format.

dwMediaMode

The media type for which the priority of the application is to be set. The value can be one LINEMEDIAMODE_Constant; only a single bit may be on. Use the value zero to set the application priority for Assisted Telephony requests.

lpExtensionID

A pointer to a structure of type LINEEXTENSIONID. This parameter gets ignored.

dwRequestMode

If the dwMediaMode parameter is zero, this parameter specifies the Assisted Telephony request mode for which priority is to be set. It must be either LINEREQUESTMODE_MAKECALL or LINEREQUESTMODE_MEDIACALL. This parameter gets ignored if dwMediaMode is nonzero.

lpszExtensionName

This parameter gets ignored.

dwPriority

The new priority for the application. If the value 0 is passed, the application gets removed from the priority list for the specified media or request mode (if it was already not present, no error gets generated). If the value 1 is passed, the application gets inserted as the highest priority application for the media or request mode (and removed from a lower-priority position, if it was already in the list). Any other value generates an error.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INIFILECORRUPT
- LINEERR_INVALREQUESTMODE
- LINEERR_INVALAPPNAME
- LINEERR_NOMEM
- LINEERR_INVALMEDIAMODE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPARAM
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALPOINTER

lineSetCallPrivilege

The lineSetCallPrivilege function sets the application privilege to the specified privilege.

Function Details

```
LONG WINAPI lineSetCallPrivilege(  HCALL hCall,
  DWORD dwCallPrivilege
);
```

Parameters

hCall

A handle to the call whose privilege is to be set. The call state of hCall can be any state.

dwCallPrivilege

The privilege that the application can have for the specified call. This parameter uses one and only one LINECALLPRIVILEGE_Constant.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCALLSTATE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCALLPRIVILEGE
- LINEERR_UNINITIALIZED
- LINEERR_NOMEM

lineSetNumRings

The lineSetNumRings function sets the number of rings that must occur before an incoming call is answered. Use this function to implement a toll saver-style function. It allows multiple, independent applications to each register the number of rings. The function lineGetNumRings returns the minimum number of rings that are requested. The application that answers incoming calls can use it to determine the number of rings that it should wait before answering the call.

Function Details

```
LONG WINAPI lineSetNumRings(  HLINE hLine,
  DWORD dwAddressID,
  DWORD dwNumRings
);
```

Parameters

hLine

A handle to the open line device.

dwAddressID

An address on the line device. An address identifier permanently associates with an address; the identifier remains constant across operating system upgrades.

dwNumRings

The number of rings before a call should be answered to honor the toll saver requests from all applications.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALADDRESSID

- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_UNINITIALIZED

lineSetStatusMessages

The lineSetStatusMessages function enables an application to specify the notification messages to receive for events that are related to status changes for the specified line or any of its addresses.

Function Details

```
LONG lineSetStatusMessages(  HLINE hLine,
  DWORD dwLineStates,
  DWORD dwAddressStates
);
```

Parameters

hLine

A handle to the line device.

dwLineStates

A bit array that identifies for which line-device status changes a message is to be sent to the application. This parameter uses the following LINEDEVSTATE_constants:

- LINEDEVSTATE_OTHER -Device-status items other than the following ones changed. The application should check the current device status to determine which items changed.
- LINEDEVSTATE_RINGING -The switch tells the line to alert the user. Service providers notify applications on each ring cycle by sending LINE_LINEDEVSTATE messages that contain this constant. For example, in the United States, service providers send a message with this constant every 6 seconds.
- LINEDEVSTATE_NUMCALLS -The number of calls on the line device changed.
- LINEDEVSTATE_REINIT -Items changed in the configuration of line devices. To become aware of these changes (as with the appearance of new line devices) the application should reinitialize its use of TAPI. New lineInitialize, lineInitializeEx, and lineOpen requests get denied until applications have shut down their usage of TAPI. The hDevice parameter of the LINE_LINEDEVSTATE message remains NULL for this state change as it applies to any lines in the system. Because of the critical nature of LINEDEVSTATE_REINIT, such messages cannot be masked, so the setting of this bit is ignored, and the messages always get delivered to the application.
- LINEDEVSTATE_REMOVED -Indicates that the service provider is removing the device from the system (most likely through user action, through a control panel or similar utility). Normally, a LINE_CLOSE message on the device immediately follows LINE_LINEDEVSTATE message with this value. Subsequent attempts to access the device prior to TAPI being reinitialized result in LINEERR_NODEVICE being returned to the application. If a service provider sends a LINE_LINEDEVSTATE message that contains this value to TAPI, TAPI passes it along to applications that have negotiated TAPI version 1.4 or later; applications that negotiate a previous TAPI version do not receive any notification.

dwAddressStates

A bit array that identifies for which address status changes a message is to be sent to the application. This parameter uses the following LINEADDRESSSTATE_constant:

- LINEADDRESSSTATE_NUMCALLS -The number of calls on the address changed. This change results from events such as a new incoming call, an outgoing call on the address, or a call changing its hold status.

lineSetTollList

The lineSetTollList function manipulates the toll list.

Function Details

```
LONG WINAPI lineSetTollList(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  LPCSTR lpszAddressIn,
  DWORD dwTollListOption
);
```

Parameters

hLineApp

The application handle that lineInitializeEx returns. If an application has not yet called the lineInitializeEx function, it can set the hLineApp parameter to NULL.

dwDeviceID

The device identifier for the line device upon which the call is intended to be dialed, so variations in dialing procedures on different lines can be applied to the translation process.

lpszAddressIn

A pointer to a null-terminated string that contains the address from which the prefix information is to be extracted for processing. Ensure that this parameter is not NULL, and also ensure that it is in the canonical address format.

dwTollListOption

The toll list operation to be performed. This parameter uses one and only one of the LINETOLLLISTOPTION_Constants.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_BADDEVICEID
- LINEERR_NODRIVER
- LINEERR_INVALAPPHANDLE
- LINEERR_NOMEM
- LINEERR_INVALADDRESS

- LINEERR_OPERATIONFAILED
- LINEERR_INVALPARAM
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INIFILECORRUPT
- LINEERR_UNINITIALIZED
- LINEERR_INVALLOCATION

lineSetupConference

The lineSetupConference function initiates a conference for an existing two-party call that the hCall parameter specifies. A conference call and consult call are established, and the handles return to the application. Use the consult call to dial the third party and the conference call replaces the initial two-party call. The application can also specify the destination address of the consult call that will allow the PBX to dial the call for the application.

Function Details

```
LONG lineSetupConference (HCALL hCall,
HLINE hLine,
LPHCALL lphConfCall,
LPHCALL lphConsultCall,
DWORD dwNumParties,
LPLINECALLPARAMS const lpCallParams
);
```

Parameters

hCall

The handle of the existing two-party call. Ensure that the application is the owner of the call.

hLine

The line on which the initial two-party call was made. This parameter is not used because hCall must be set.

lphConfCall

A pointer to the conference call handle. The service provider allocates this call and returns the handle to the application.

lphConsultCall

A pointer to the consult call. If the application does not specify the destination address in the call parameters, it should use this call handle to dial the consult call. If the destination address is specified, the consult call will be made using this handle.

dwNumParties

The number of parties in the conference call. Currently the Cisco Unified TAPI Service Provider supports a three-party conference call.

lpCallParams

The call parameters that are used to set up the consult call. The application can specify the destination address if it wants the consult call to be dialed for it in the conference setup.

lineSetupTransfer

The lineSetupTransfer function initiates a transfer of the call that the hCall parameter specifies. It establishes a consultation call, lphConsultCall, on which the party can be dialed that can become the destination of the transfer. The application acquires owner privilege to the lphConsultCall parameter.

Function Details

```
LONG lineSetupTransfer(  HCALL hCall,
  LPHCALL lphConsultCall,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

hCall

The handle of the call to be transferred. Ensure that the application is an owner of the call and ensure that the call state of hCall is connected.

lphConsultCall

A pointer to an hCall handle. This location is then loaded with a handle that identifies the temporary consultation call. When setting up a call for transfer, a consultation call automatically gets allocated that enables lineDial to dial the address that is associated with the new transfer destination of the call. The originating party can carry on a conversation over this consultation call prior to completing the transfer. The call state of hConsultCall does not apply.

This transfer procedure may not be valid for some line devices. The application may need to ignore the new consultation call and remove the hold on an existing held call (using lineUnhold) to identify the destination of the transfer. On switches that support cross-address call transfer, the consultation call can exist on a different address than the call that is to be transferred. It may also be necessary to set up the consultation call as an entirely new call, by lineMakeCall, to the destination of the transfer. The address capabilities of the call specifies which forms of transfer are available.

lpCallParams

The dwNoAnswerTimeout attribute of the lpCallParams field is checked and, if is non-zero, used to automatically disconnect a call if it is not answered after the specified time.

lineShutdown

The lineShutdown function shuts down the usage of the line abstraction of the API.

Function Details

```
LONG lineShutdown(  HLINEAPP hLineApp
);
```

Parameters

hLineApp

The usage handle of the application for the line API.

lineTranslateAddress

The lineTranslateAddress function translates the specified address into another format.

Function Details

```
LONG WINAPI lineTranslateAddress(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAPIVersion,
  LPCSTR lpszAddressIn,
  DWORD dwCard,
  DWORD dwTranslateOptions,
  LPLINETRANSLATEOUTPUT lpTranslateOutput
);
```

Parameters

hLineApp

The application handle that lineInitializeEx returns. If a TAPI 2.0 application has not yet called the lineInitializeEx function, it can set the hLineApp parameter to NULL. TAPI 1.4 applications must still call lineInitialize first.

dwDeviceID

The device identifier for the line device upon which the call is intended to be dialed, so variations in dialing procedures on different lines can be applied to the translation process.

dwAPIVersion

Indicates the highest version of TAPI that the application supports (not necessarily the value that is negotiated by lineNegotiateAPIVersion on some particular line device).

lpszAddressIn

Pointer to a null-terminated string that contains the address from which the information is to be extracted for translation. This parameter must either use the canonical address format or an arbitrary string of dialable digits (non-canonical). This parameter must not be NULL. If the AddressIn contains a subaddress or name field, or additional addresses separated from the first address by CR and LF characters, only the first address gets translated.

dwCard

The credit card to be used for dialing. This parameter proves valid only if the CARDOVERRIDE bit is set in dwTranslateOptions. This parameter specifies the permanent identifier of a Card entry in the [Cards] section in the registry (as obtained from lineTranslateCaps) that should be used instead of the PreferredCardID that is specified in the definition of the CurrentLocation. It does not cause the PreferredCardID parameter of the current Location entry in the registry to be modified; the override applies only to the current translation operation. This parameter gets ignored if the CARDOVERRIDE bit is not set in dwTranslateOptions.

dwTranslateOptions

The associated operations to be performed prior to the translation of the address into a dialable string. This parameter uses one of the LINETRANSLATEOPTION_Constants.

Note If you have set the LINETRANSLATEOPTION_CANCELCALLWAITING bit, also set the LINECALLPARAMFLAGS_SECURE bit in the dwCallParamFlags member of the LINECALLPARAMS structure (passed in to lineMakeCall through the lpCallParams parameter). This action prevents the line device from using dialable digits to suppress call interrupts.

lpTranslateOutput

A pointer to an application-allocated memory area to contain the output of the translation operation, of type LINETRANSLATEOUTPUT. Prior to calling lineTranslateAddress, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_BADDEVICEID
- LINEERR_INVALPOINTER
- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_NODRIVER
- LINEERR_INIFILECORRUPT
- LINEERR_NOMEM
- LINEERR_INVALADDRESS
- LINEERR_OPERATIONFAILED
- LINEERR_INVALAPPHANDLE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCARD
- LINEERR_STRUCTURETOOSMALL
- LINEERR_INVALPARAM

lineTranslateDialog

The lineTranslateDialog function displays an application-modal dialog box that allows the user to change the current location of a phone number that is about to be dialed, adjust location and calling card parameters, and see the effect.

Function Details

```
LONG WINAPI lineTranslateDialog(  HLINEAPP hLineApp,
  DWORD dwDeviceID,
  DWORD dwAPIVersion,
  HWND hwndOwner,
  LPCSTR lpszAddressIn
);
```

Parameters

hLineApp

The application handle that lineInitializeEx returns. If an application has not yet called the lineInitializeEx function, it can set the hLineApp parameter to NULL.

dwDeviceID

The device identifier for the line device upon which the call is intended to be dialed, so variations in dialing procedures on different lines can be applied to the translation process.

dwAPIVersion

Indicates the highest version of TAPI that the application supports (not necessarily the value that lineNegotiateAPIVersion negotiates on the line device that dwDeviceID indicates).

hwndOwner

A handle to a window to which the dialog box is to be attached. Can be a NULL value to indicate that any window that is created during the function should have no owner window.

lpszAddressIn

A pointer to a null-terminated string that contains a phone number that is used, in the lower portion of the dialog box, to show the effect of the user's changes on the location parameters. Ensure that the number is in canonical format; if noncanonical, the phone number portion of the dialog box does not display. You can leave this pointer NULL, in which case the phone number portion of the dialog box does not display. If the lpszAddressIn parameter contains a subaddress or name field, or additional addresses separated from the first address by CR and LF characters, only the first address gets used in the dialog box.

Return Values

Returns zero if request succeeds or a negative number if an error occurs. Possible return values follow:

- LINEERR_BADDEVICEID
- LINEERR_INVALPARAM
- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_INVALPOINTER
- LINEERR_INIFILECORRUPT
- LINEERR_NODRIVER
- LINEERR_INUSE
- LINEERR_NOMEM

- LINEERR_INVALADDRESS
- LINEERR_INVALAPPHANDLE
- LINEERR_OPERATIONFAILED

lineUnhold

The lineUnhold function retrieves the specified held call.

Function Details

```
LONG lineUnhold(  HCALL hCall
);
```

Parameters

hCall

The handle to the call to be retrieved. The application must be an owner of this call. The call state of hCall must be onHold, onHoldPendingTransfer, or onHoldPendingConference.

lineUnpark

The lineUnpark function retrieves the call that is parked at the specified address and returns a call handle for it.

Function Details

```
LONG WINAPI lineUnpark(HLINE hLine,
DWORD dwAddressID,
LPHCALL lphCall,
LPCSTR lpszDestAddress
);
```

Parameters

hLine

Handle to the open line device on which a call is to be unparked.

dwAddressID

Address on hLine at which the unpark is to be originated. An address identifier permanently associates with an address; the identifier remains constant across operating system upgrades.

lphCall

Pointer to the location of type HCALL where the handle to the unparked call is returned. This handle is unrelated to any other handle that previously may have been associated with the retrieved call, such as the handle that might have been associated with the call when it was originally parked. The application acts as the initial sole owner of this call.

lpszDestAddress

Pointer to a null-terminated character buffer that contains the address where the call is parked. The address displays in standard dialable address format.

TAPI Line Messages

This section describes the line messages that the Cisco Unified TSP supports. These messages notify the application of asynchronous events such as a new call arriving in the Cisco Unified Communications Manager. The messages get sent to the application by the method that the application specifies in lineInitializeEx

.

Table 13: TAPI Line Messages

TAPI Line Messages
LINE_ADDRESSSTATE, on page 196
LINE_APPNEWCALL, on page 197
LINE_CALLDEVSPECIFIC, on page 198
LINE_CALLINFO, on page 198
LINE_CALLSTATE, on page 199
LINE_CLOSE, on page 203
LINE_CREATE, on page 203
LINE_DEVSPECIFIC, on page 204
LINE_DEVSPECIFICFEATURE, on page 205
LINE_GATHERDIGITS, on page 206
LINE_GENERATE, on page 207
LINE_LINEDEVSTATE, on page 208
LINE_MONITORDIGITS, on page 209
LINE_MONITORTONE, on page 209
LINE_REMOVE, on page 210
LINE_REPLY, on page 211
LINE_REQUEST, on page 212

LINE_ADDRESSSTATE

The LINE_ADDRESSSTATE message gets sent when the status of an address changes on a line that is currently open by the application. The application can invoke lineGetAddressStatus to determine the current status of the address.

Function Details

```
LINE_ADDRESSSTATE
dwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) idAddress;
dwParam2 = (DWORD) AddressState;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

A handle to the line device.

dwCallbackInstance

The callback instance supplied when the line is opened.

dwParam1

The address identifier of the address that changed status.

dwParam2

The address state that changed. Can be a combination of these values:

LINEADDRESSSTATE_OTHER

Address-status items other than those that are in the following list changed. The application should check the current address status to determine which items changed.

LINEADDRESSSTATE_DEVSPECIFIC

The device-specific item of the address status changed.

LINEADDRESSSTATE_INUSEZERO

The address changed to idle (it is now in use by zero stations).

LINEADDRESSSTATE_INUSEONE

The address changed from idle or from being used by many bridged stations to being used by just one station.

LINEADDRESSSTATE_INUSEMANY

The monitored or bridged address changed from being used by one station to being used by more than one station.

LINEADDRESSSTATE_NUMCALLS

The number of calls on the address has changed. This change results from events such as a new inbound call, an outbound call on the address, or a call changing its hold status.

LINEADDRESSSTATE_FORWARD

The forwarding status of the address changed, including the number of rings for determining a no-answer condition. The application should check the address status to determine details about the current forwarding status of the address.

LINEADDRESSSTATE_TERMINALS

The terminal settings for the address changed.

LINEADDRESSSTATE_CAPSCHANGE

Indicates that due to configuration changes that the user made, or other circumstances, one or more of the members in the LINEADDRESSCAPS structure for the address changed. The application should use lineGetAddressCaps to read the updated structure. Applications that support API versions earlier than 1.4 receive a LINEDEVSTATE_REINIT message that requires them to shut down and reinitialize their connection to TAPI to obtain the updated information.

dwParam3

This parameter is not used.

LINE_APPNEWCALL

The LINE_APPNEWCALL message informs an application when a new call handle is spontaneously created on its behalf (other than through an API call from the application, in which case the handle would have been returned through a pointer parameter that passed into the function).

Function Details

```
LINE_APPNEWCALL
dwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) dwInstanceData;
dwParam1 = (DWORD) dwAddressID;
dwParam2 = (DWORD) hCall;
dwParam3 = (DWORD) dwPrivilege;
```

Parameters

dwDevice

The handle of the application to the line device on which the call was created.

dwCallbackInstance

The callback instance that is supplied when the line belonging to the call is opened.

dwParam1

Identifier of the address on the line on which the call appears.

dwParam2

The handle of the application to the new call.

dwParam3

The privilege of the application to the new call (LINECALLPRIVILEGE_OWNER or LINECALLPRIVILEGE_MONITOR).

LINE_CALLDEVSPECIFIC

The TSPI LINE_CALLDEVSPECIFIC message is sent to notify TAPI about device-specific events that occur on a call. The meaning of the message and the interpretation of the dwParam1 through dwParam3 parameters are device specific.

Function Details

```
LINE_CALLDEVSPECIFIC
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) hCallDevice;
dwMsg = (DWORD) LINE_CALLDEVSPECIFIC;
dwParam1 = (DWORD) DeviceData1;
dwParam2 = (DWORD) DeviceData2;
dwParam3 = (DWORD) DeviceData3;
```

Parameters

htLine

The TAPI opaque object handle to the line device.

htCall

The TAPI opaque object handle to the call device.

dwMsg

The value LINE_CALLDEVSPECIFIC.

dwParam1

Device specific

dwParam2

Device specific

dwParam3

Device specific

LINE_CALLINFO

The TAPI LINE_CALLINFO message gets sent when the call information about the specified call has changed. The application can invoke lineGetCallInfo to determine the current call information.

Function Details

```
LINE_CALLINFO
hDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) CallInfoState;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hDevice

A handle to the call.

dwCallbackInstance

The callback instance that is supplied when the call's line is opened.

dwParam1

The call information item that changed. Can be one or more of the

LINECALLINFOSTATE_constants.

dwParam2

This parameter is not used.

dwParam3

This parameter is not used.

LINE_CALLSTATE

The LINE_CALLSTATE message gets sent when the status of the specified call changes. Typically, several such messages occur during the lifetime of a call. Applications get notified of new incoming calls with this message; the new call exists in the offering state. The application can use the lineGetCallStatus function to retrieve more detailed information about the current status of the call.

Function Details

```
LINE_CALLSTATE
dwDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) CallState;
dwParam2 = (DWORD) CallStateDetail;
dwParam3 = (DWORD) CallPrivilege;
```

Parameters

dwDevice

A handle to the call.

dwCallbackInstance

The callback instance that is supplied when the line that belongs to this call is opened.

dwParam1

The new call state. Cisco Unified TSP supports only the following LINECALLSTATE_values:

LINECALLSTATE_IDLE

The call remains idle; no call actually exists.

LINECALLSTATE_OFFERING

The call gets offered to the station, which signals the arrival of a new call. In some environments, a call in the offering state does not automatically alert the user. The switch that instructs the line to ring does alerts; it does not affect any call states.

LINECALLSTATE_ACCEPTED

The system offered the call and it has been accepted. This indicates to other (monitoring) applications that the current owner application claimed responsibility for answering the call. In ISDN, this also indicates that alerting to both parties started.

LINECALLSTATE_CONFERENCED

The call is a member of a conference call and is logically in the connected state.

LINECALLSTATE_DIALTONE

The call receives a dial tone from the switch, which means that the switch is ready to receive a dialed number.

LINECALLSTATE_DIALING

Destination address information (a phone number) is sent to the switch over the call. The lineGenerateDigits does not place the line into the dialing state.

LINECALLSTATE_RINGBACK

The call receives ringback from the called address. Ringback indicates that the call has reached the other station and is being alerted.

LINECALLSTATE_ONHOLDPENDCONF

The call currently remains on hold while it gets added to a conference.

LINECALLSTATE_CONNECTED

The call is established and the connection is made. Information can flow over the call between the originating address and the destination address.

LINECALLSTATE_PROCEEDING

Dialing completes and the call proceeds through the switch or telephone network.

LINECALLSTATE_ONHOLD

The switch keeps the call on hold.

LINECALLSTATE_ONHOLDPENDTRANSFER

The call that is currently on hold awaits transfer to another number.

LINECALLSTATE_DISCONNECTED

The remote party disconnected from the call.

LINECALLSTATE_UNKNOWN

The state of the call is not known. This state may occur due to limitations of the call-progress detection implementation.

Cisco Unified TSP supports two new call states that indicate more information about the call state within the Cisco Unified Communications Manager setup. The standard TAPI call state is set to

LINECALLSTATE_UNKNOWN and the following call states will be ORed with the unknown call state.

#define CLDSMT_CALL_PROGRESSING_STATE 0x0100000

The Progressing state indicates that the call is in progress over the network. The application must negotiate extension version 0x00050001 to receive this call state.

#define CLDSMT_CALL_WAITING_STATE 0x02000000

The waiting state indicates that the REFER request is in progress on Referrer's line and the application should not request any other function on this call. All the requests will result in LINEERR_INVALCALLSTATE. Application has to negotiate extension version 0x00070000 to receive this call state.

#define CLDSMT_CALL_WHISPER_STATE 0x03000000

The whisper state indicates that the Intercom call is connected in one-way audio mode. The Intercom originator cannot issue other function other that to drop the Intercom call. While at destination side, the system allows only Talkback and dropping call. All other requests result in LINEERR_OPERATIONUNAVAIL.

dwParam2

Call-state-dependent information.

- If dwParam1 is LINECALLSTATE_CONNECTED, dwParam2 contains details about the connected mode. This parameter uses the following LINECONNECTEDMODE_constants:

  LINECONNECTEDMODE_ACTIVE

  Call connects at the current station (the current station acts as a participant in the call).

  LINECONNECTEDMODE_INACTIVE

  Call stays active at one or more other stations, but the current station does not participate in the call.

  When a call is disconnected with cause code = DISCONNECTMODE_TEMPFAILURE and the lineState = LINEDEVSTATE_INSERVICE, applications must take care of dropping the call. If the application terminates media for a device, then it is also takes the responsibility to stop the RTP streams for the same call. Cisco Unified TSP will not provide Stop Transmission/Reception events to applications in this scenario. The behavior is exactly the same with IP phones. The user must hang up the disconnected -temp fail call on IP phone to stop the media. The application is also responsible for stopping the RTP streams in case the line goes out of service (LINEDEVSTATE_OUTOFSERVICE) and the call on a line is reported as IDLE.

  Note If an application with negotiated extension version 0x00050001 or greater receives device-specific CLDSMT_CALL_PROGRESSING_STATE = 0x01000000 with LINECALLSTATE_UNKNOWN, the cause code is reported as the standard Q931 cause codes in dwParam2.

- If dwParam1 specifies LINECALLSTATE_DIALTONE, dwParam2 contains the details about the dial tone mode. This parameter uses the following LINEDIALTONEMODE_constant:

  LINEDIALTONEMODE_UNAVAIL

  The dial tone mode is unavailable and cannot become known.

- If dwParam1 specifies LINECALLSTATE_OFFERING, dwParam2 contains details about the connected mode. This parameter uses the following LINEOFFERINGMODE_constants:

  LINEOFFERINGMODE_ACTIVE

  The call alerts at the current station (accompanied by LINEDEVSTATE_RINGING messages) and, if an application is set up to automatically answer, it answers. For TAPI versions 1.4 and later, if the call state mode is ZERO, the application assumes that the value is active (which represents the situation on a non-bridged address).

  Note The Cisco Unified TSP does not send LINEDEVSTATE_RINGING messages until the call is accepted and moves to the LINECALLSTATE_ACCEPTED state. IP_phones auto-accept calls. CTI ports and CTI route points do not auto-accept calls. Call the lineAccept() function to accept the call at these types of devices.

- If dwParam1 specifies LINECALLSTATE_DISCONNECTED, dwParam2 contains details about the disconnect mode. This parameter uses the following LINEDISCONNECTMODE_constants:

  LINEDISCONNECTMODE_NORMAL

  This specifies a normal disconnect request by the remote party; call terminated normally.

  LINEDISCONNECTMODE_UNKNOWN

  The reason for the disconnect request remains unknown.

  LINEDISCONNECTMODE_REJECT

  The remote user rejected the call.

  LINEDISCONNECTMODE_BUSY

  The station that belongs to the remote user is busy.

  LINEDISCONNECTMODE_NOANSWER

  The station that belongs to the remote user does not answer.

  LINEDISCONNECTMODE_CONGESTION

  This message indicates that the network is congested.

  LINEDISCONNECTMODE_UNAVAIL

  The reason for the disconnect remains unavailable and cannot become known later.

  LINEDISCONNECTMODE_FACCMC

  Indicates that the FAC/CMC feature disconnected the call.

  Note LINEDISCONNECTMODE_FACCMC is returned only if the extension version that is negotiated on the line is 0x00050000 (6.0(1)) or higher. If the negotiated extension version is not at least 0x00050000, TSP sets the disconnect mode to LINEDISCONNECTMODE_UNAVAIL.

dwParam3

If zero, this parameter indicates that no change in the privilege occurred for the call to this application.

If nonzero, this parameter specifies the privilege for the application to the call. This occurs in the following situations: (1) The first time that the application receives a handle to this call; (2) When the application is the target of a call hand-off (even if the application already was an owner of the call). This parameter uses the following LINECALLPRIVILEGE_ constants:

LINECALLPRIVILEGE_MONITOR

The application has monitor privilege.

LINECALLPRIVILEGE_OWNER

The application has owner privilege.

LINE_CLOSE

The LINE_CLOSE message gets sent when the specified line device has been forcibly closed. The line device handle or any call handles for calls on the line no longer remains valid after this message is sent.

Function Details

```
LINE_CLOSE
dwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) 0;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

A handle to the line device that was closed. This handle is no longer valid

dwCallbackInstance

The callback instance that is supplied when the line that belongs to this call is opened.

dwParam1

This parameter is not used.

dwParam2

This parameter is not used.

dwParam3

This parameter is not used.

LINE_CREATE

The LINE_CREATE message informs the application of the creation of a new line device.

Note CTI Manager cluster support, extension mobility, change notification, and user addition to the directory can generate LINE_CREATE events.

Function Details

```
LINE_CREATE
dwDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) idDevice;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

This parameter is not used.

dwCallbackInstance

This parameter is not used.

dwParam1

The dwDeviceID of the newly created device.

dwParam2

This parameter is not used.

dwParam3

This parameter is not used.

LINE_DEVSPECIFIC

The LINE_DEVSPECIFIC message notifies the application about device-specific events that occur on a line, address, or call. The meaning of the message and interpretation of the parameters are device specific.

Function Details

```
LINE_DEVSPECIFIC
dwDevice = (DWORD) hLineOrCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) DeviceSpecific1;
dwParam2 = (DWORD) DeviceSpecific2;
dwParam3 = (DWORD) DeviceSpecific3;
```

Parameters

dwDevice

This device-specific parameter specifies a handle to either a line device or call.

dwCallbackInstance

The callback instance that is supplied when the line opens.

dwParam1

is device specific

dwParam2

is device specific

dwParam3

is device specific

LINE_DEVSPECIFICFEATURE

This line message, added in Cisco Unified Communications Manager Release 6.0, enables a Do Not Disturb (DND) change notification event. Cisco TSP notifies applications by using the LINE_DEVSPECIFICFEATURE message about changes in the DND configuration or status. In order to receive change notifications an application needs to enable DEVSPECIFIC_DONOTDISTURB_CHANGED message flag by using lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

LINE_DEVSPECIFICFEATURE message is sent to notify the application about device-specific events occurring on a line device. In case of a DND change notification, the message includes information about the type of change that occurred on a device and resulted feature status or configured option.

Function Details

```
dwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) PHONEBUTTONFUNCTION_DONOTDISTURB;
dwParam2 = (DWORD) typeOfChange;
dwParam3 = (DWORD) currentValue;

enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE      = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT    = 2
};

enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN  = 0,
    DoNotDisturbStatus_ENABLED  = 1,
    DoNotDisturbStatus_DISABLED = 2
};

enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED  = 1,
    DoNotDisturb_OPTION_CHANGED  = 2
};
```

Parameters

dwDevice

A handle to a line device.

dwCallbackInstance

The callback instance supplied when opening the line.

dwParam1

Always equal to PHONEBUTTONFUNCTION_DONOTDISTURB for the Do-Not-Disturb change notification

dwParam2

Indicates the type of change and can have one of the following enum values:

```
enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED  = 1,
    DoNotDisturb_OPTION_CHANGED  = 2
};
```

dwParam3

If the dwParm2 indicates status change (is equal to DoNotDisturb_STATUS_CHANGED) this parameter can have one of the following enum values:

```
enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN  = 0,
    DoNotDisturbStatus_ENABLED  = 1,
    DoNotDisturbStatus_DISABLED = 2
};
```

If the dwParm2 indicates option change (is equal to DoNotDisturb_OPTION_CHANGED) this parameter can have one of the following enum values:

```
enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE      = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT    = 2
};
```

LINE_GATHERDIGITS

The TAPI LINE_GATHERDIGITS message is sent when the current buffered digit-gathering request is terminated or canceled. You can examine the digit buffer after the application receives this message.

Function Details

```
LINE_GATHERDIGITS
hDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) GatherTermination;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hDevice

A handle to the call.

dwCallbackInstance

The callback instance that is supplied when the line opens.

dwParam1

The reason why digit gathering terminated. This parameter must be one and only one of the LINEGATHERTERM_constants.

dwParam2

Unused.

dwParam3

The tick count (number of milliseconds since Windows started) at which the digit gathering completes. For TAPI versions earlier than 2.0, this parameter is not used.

LINE_GENERATE

The TAPI LINE_GENERATE message notifies the application that the current digit or tone generation terminated. Only one such generation request can be in progress an a given call at any time. This message also gets sent when digit or tone generation is canceled.

Function Details

```
LINE_GENERATE
hDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) GenerateTermination;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hDevice

A handle to the call.

dwCallbackInstance

The callback instance that is supplied when the line opens.

dwParam1

The reason that digit or tone generation terminates. This parameter must be the only one of the LINEGENERATETERM_constants.

dwParam2

This parameter is not used.

dwParam3

The tick count (number of milliseconds since Windows started) at which the digit or tone generation completes. For API versions earlier than 2.0, this parameter is not used.

LINE_LINEDEVSTATE

The TAPI LINE_LINEDEVSTATE message gets sent when the state of a line device changes. The application can invoke lineGetLineDevStatus to determine the new status of the line.

Function Details

```
LINE_LINEDEVSTATE
hDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) DeviceState;
dwParam2 = (DWORD) DeviceStateDetail1;
dwParam3 = (DWORD) DeviceStateDetail2;
```

Parameters

hDevice

A handle to the line device. This parameter is NULL when dwParam1 is LINEDEVSTATE_REINIT.

dwCallbackInstance

The callback instance that is supplied when the line is opened. If the dwParam1 parameter is LINEDEVSTATE_REINIT, the dwCallbackInstance parameter is not valid and is set to zero.

dwParam1

The line device status item that changed. The parameter can be one or more of the LINEDEVSTATE_constants.

LINEDEVSTATE_OUTOFSERVICE

Indicates the line device unregisters as it enters Energywise DeepSleep/PowersavePlus mode

dwParam2

The interpretation of this parameter depends on the value of dwParam1. If dwParam1 is LINEDEVSTATE_RINGING, dwParam2 contains the ring mode with which the switch instructs the line to ring. Valid ring modes include numbers in the range one to dwNumRingModes, where dwNumRingModes specifies a line device capability.

If dwParam1 is LINEDEVSTATE_REINIT, and TAPI issued the message as a result of translation of a new API message into a REINIT message, dwParam2 contains the dwMsg parameter of the original message (for example, LINE_CREATE or LINE_LINEDEVSTATE). If dwParam2 is zero, this indicates that the REINIT message represents a real REINIT message that requires the application to call lineShutdown at its earliest convenience.

If dw Param1 is LINEDEVSTATE_OUTOFSERVICE, dwParam2 contains the reason EnergyWisePowerSavePlus when the line device unregisters as it enters EnergywiseDeepSleep.

dwParam3

The interpretation of this parameter depends on the value of dwParam1. If dwParam1 is LINEDEVSTATE_RINGING, dwParam3 contains the ring count for this ring event. The ring count starts at zero.

If dwParam1 is LINEDEVSTATE_REINIT, and TAPI issued the message as a result of translation of a new API message into a REINIT message, dwParam3 contains the dwParam1 parameter of the original message (for example, LINEDEVSTATE_TRANSLATECHANGE or some other

LINEDEVSTATE_value, if dwParam2 is LINE_LINEDEVSTATE, or the new device identifier, if dwParam2 is LINE_CREATE).

LINE_MONITORDIGITS

The LINE_MONITORDIGITS message gets sent when a digit is detected. The lineMonitorDigits function controls the sending of this message.

Function Details

```
LINE_MONITORDIGITS
dwDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) Digit;
dwParam2 = (DWORD) DigitMode;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

A handle to the call.

dwCallbackInstance

The callback instance that is supplied when the line for this call is opened.

dwParam1

The low-order byte contains the last digit that is received in ASCII.

dwParam2

The digit mode that was detected. This parameter must be one and only one of the following LINEDIGITMODE_constant:

LINEDIGITMODE_DTMF

Detect digits as DTMF tones. Valid digits for DTMF include ‘0’ through ‘9’, ‘*’, and ‘#’.

dwParam3

The “tick count” (number of milliseconds after Windows started) at which the specified digit was detected. For API versions earlier than 2.0, this parameter is not used.

LINE_MONITORTONE

The LINE_MONITORTONE message gets sent when a tone is detected. The lineMonitorTones function controls the sending of this message.

Note Cisco Unified TSP supports only silent detection through LINE_MONITORTONE.

Function Details

```
LINE_MONITORTONE
dwDevice = (DWORD) hCall;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) dwAppSpecific;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) tick count;
```

Parameters

dwDevice

A handle to the call.

dwCallbackInstance

The callback instance supplied when the line opens for this call.

dwParam1

The application-specific dwAppSpecific member of the LINE_MONITORTONE structure for the tone that was detected.

dwParam2

This parameter is not used.

dwParam3

The "tick count" (number of milliseconds after Windows started) at which the specified digit was detected.

LINE_REMOVE

The LINE_REMOVE message informs an application of the removal (deletion from the system) of a line device. Generally, this parameter is not used for temporary removals, such as extraction of PCMCIA devices, but only for permanent removals in which, the service provider would no longer report the device, if TAPI were reinitialized.

Note CTI Manager cluster support, extension mobility, change notification, and user deletion from the directory can generate LINE_REMOVE events.

Function Details

```
LINE_REMOVE
dwDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) dwDeviceID;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

Reserved. Set to zero.

dwCallbackInstance

Reserved. Set to zero.

dwParam1

Identifier of the line device that was removed.

dwParam2

Reserved. Set to zero.

dwParam3

Reserved. Set to zero.

LINE_REPLY

The LINE_REPLY message reports the results of function calls that completed asynchronously.

Function Details

```
LINE_REPLY
dwDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) idRequest;
dwParam2 = (DWORD) Status;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice

This parameter is not used.

dwCallbackInstance

Returns the callback instance for this application.

dwParam1

The request identifier for which this is the reply.

dwParam2

The success or error indication. The application should cast this parameter into a long integer:

- Zero indicates success.
- A negative number indicates an error.

dwParam3

This parameter is not used.

LINE_REQUEST

The TAPI LINE_REQUEST message reports the arrival of a new request from another application.

Function Details

```
LINE_REQUEST
hDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) hRegistration;
dwParam1 = (DWORD) RequestMode;
dwParam2 = (DWORD) RequestModeDetail1;
dwParam3 = (DWORD) RequestModeDetail2;
```

Parameters

hDevice

This parameter is not used.

dwCallbackInstance

The registration instance of the application that is specified on lineRegisterRequestRecipient.

dwParam1

The request mode of the newly pending request. This parameter uses the LINEREQUESTMODE_constants.

dwParam2

If dwParam1 is set to LINEREQUESTMODE_DROP, dwParam2 contains the hWnd of the application that requests the drop. Otherwise, dwParam2 is not used.

dwParam3

If dwParam1 is set to LINEREQUESTMODE_DROP, the low-order word of dwParam3 contains the wRequestID as specified by the application requesting the drop. Otherwise, dwParam3 is not used.

TAPI Line Device Structures

The following table lists the TAPI line device structures that the Cisco Unified TSP supports. This section lists the possible values for the structure members as set by the TSP, and provides a cross reference to the functions that use them. If the value of a structure member is device, line, or call specific, the system notes the value for each condition.

Table 14: TAPI Line Device Structures

TAPI Line Device Structures
LINEADDRESSCAPS, on page 213
LINEADDRESSSTATUS, on page 224
LINEAPPINFO, on page 225
LINECALLINFO, on page 227

TAPI Line Device Structures
LINECALLLIST, on page 235
LINECALLPARAMS, on page 236
LINECALLSTATUS, on page 238
LINECARDENTRY, on page 244
LINECOUNTRYENTRY, on page 246
LINECOUNTRYLIST, on page 247
LINEDEVCAPS, on page 248
LINEDEVSTATUS, on page 253
LINEEXTENSIONID, on page 255
LINEFORWARD, on page 255
LINEFORWARDLIST, on page 259
LINEGENERATETONE, on page 259
LINEINITIALIZEEXPARAMS, on page 260
LINELOCATIONENTRY, on page 261
LINEMESSAGE, on page 263
LINEMONITORTONE, on page 264
LINEPROVIDERENTRY, on page 265
LINEPROVIDERLIST, on page 265
LINEREQMAKECALL, on page 266
LINETRANSLATECAPS, on page 267
LINETRANSLATEOUTPUT, on page 268

LINEADDRESSCAPS

Members

Members	Values
dwLineDeviceID	For All Devices:The device identifier of the line device with which this address is associated.

Members	Values
dwAddressSizedwAddressOffset	For All Devices:The size, in bytes, of the variably sized address field and the offset, in bytes, from the beginning of this data structure
dwDevSpecificSize dwDevSpecificOffset	For All Devices:0
dwAddressSharing	For All Devices:0
dwAddressStates	For All Devices (except Park DNs):LINEADDRESSSTATE_FORWARD
	For Park DNs:0

Members	Values
dwCallInfoStates	For All Devices (except Park DNs):LINECALLINFOSTATE_CALLEDID LINECALLINFOSTATE_CALLERID LINECALLINFOSTATE_CALLID LINECALLINFOSTATE_CONNECTEDID LINECALLINFOSTATE_MEDIAMODE LINECALLINFOSTATE_MONITORMODES LINECALLINFOSTATE_NUMMONITORS LINECALLINFOSTATE_NUMOWNERDECR LINECALLINFOSTATE_NUMOWNERINCR LINECALLINFOSTATE_ORIGIN LINECALLINFOSTATE_REASON LINECALLINFOSTATE_REDIRECTINGID LINECALLINFOSTATE_REDIRECTIONID
	For Park DNs: LINECALLINFOSTATE_CALLEDID LINECALLINFOSTATE_CALLERID LINECALLINFOSTATE_CALLID LINECALLINFOSTATE_CONNECTEDID LINECALLINFOSTATE_NUMMONITORS LINECALLINFOSTATE_NUMOWNERDECR LINECALLINFOSTATE_NUMOWNERINCR LINECALLINFOSTATE_ORIGIN LINECALLINFOSTATE_REASON LINECALLINFOSTATE_REDIRECTINGID LINECALLINFOSTATE_REDIRECTIONID
dwCallerIDFlags	For All Devices:LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED

Members	Values
dwCalledIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN
dwConnectedIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED
dwRedirectionIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED
dwRedirectingIDFlags	For All Devices:LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN

Members	Values
dwCallStates	For IP Phones and CTI Ports:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DIALING LINECALLSTATE_DIALTONE LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_ONHOLDPENDCONF LINECALLSTATE_ONHOLDPENDTRANSFER LINECALLSTATE_PROCEEDING LINECALLSTATE_RINGBACK LINECALLSTATE_UNKNOWN
	For CTI Route Points (without media):LINECALLSTATE_ACCEPTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_UNKNOWN For CTI Route Points (with media):LINECALLSTATE_ACCEPTED LINECALLSTATE_CONNECTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_ONHOLD LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_UNKNOWN

Members	Values
	For Park DNs:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_UNKNOWN
dwDialToneModes	For IP Phones and CTI Ports:LINEDIALTONEMODE_UNAVAIL
	For CTI Route Points and Park DNs:0
dwBusyModes	For All Devices:0
dwSpecialInfo	For All Devices:0
dwDisconnectModes	For All Devices:LINEDISCONNECTMODE_BADDADDRESS LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_TEMPFAILURE LINEDISCONNECTMODE_UNREACHABLE LINEDISCONNECTMODE_FACCMC (if negotiated extension version is 0x00050000 or greater)
dwMaxNumActiveCalls	For IP Phones, CTI Ports, and Park DNs:1
	For CTI Route Points (without media): 0 For CTI Route Points (with media):Cisco Unified Communications Manager Administration configuration

Members	Values
dwMaxNumOnHoldCalls	For IP Phones, CTI Ports:200
	For CTI Route Points:0 For CTI Route Points (with media):Cisco Unified Communications Manager Administration configuration (same configuration as dwMaxNumActiveCalls)
	For Park DNs: 1
dwMaxNumOnHoldPendingCalls	For IP Phones and CTI Ports:1
	For CTI Route Points and Park DNs:0
dwMaxNumConference	For IP Phones, CTI Ports, and Park DNs:16
	For CTI Route Points:0
dwMaxNumTransConf	For All Devices:0
dwAddrCapFlags	For IP Phones:LINEADDRCAPFLAGS_CONFERENCEHELD LINEADDRCAPFLAGS_DIALED LINEADDRCAPFLAGS_FWDSTATUSVALID LINEADDRCAPFLAGS_PARTIALDIAL LINEADDRCAPFLAGS_TRANSFERHELD
	For CTI Ports:LINEADDRCAPFLAGS_CONFERENCEHELD LINEADDRCAPFLAGS_DIALED LINEADDRCAPFLAGS_ACCEPTTOALERT LINEADDRCAPFLAGS_FWDSTATUSVALID LINEADDRCAPFLAGS_PARTIALDIAL LINEADDRCAPFLAGS_TRANSFERHELD
	For CTI Route Points:LINEADDRCAPFLAGS_ACCEPTTOALERT LINEADDRCAPFLAGS_FWDSTATUSVALID LINEADDRCAPFLAGS_ROUTEPOINT
	For Park DNs:LINEADDRCAPFLAGS_NOEXTERNALCALLS LINEADDRCAPFLAGS_NOINTERNALCALLS

Members	Values
dwCallFeatures	For IP Phones (except VG248 and ATA186) and CTI Ports:LINECALLFEATURE_ACCEPT LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_ANSWER LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_PARK LINECALLFEATURE_PREPAREADDTOCONF LINECALLFEATURE_REDIRECT LINECALLFEATURE_SETUPCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_UNHOLD LINECALLFEATURE_UNPARK

Members	Values
	For VG248 and ATA186 Devices:LINECALLFEATURE_ACCEPT LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_PARK LINECALLFEATURE_PREPAREADDTOCONF LINECALLFEATURE_REDIRECT LINECALLFEATURE_SETUPCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_UNHOLD INECALLFEATURE_UNPARK

Members	Values
dwCallFeatures (continued)	For CTI Route Points (without media):LINECALLFEATURE_ACCEPT LINECALLFEATURE_DROP LINECALLFEATURE_REDIRECT For CTI Route Points (with media):LINECALLFEATURE_ACCEPT LINECALLFEATURE_ANSWER LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_REDIRECT LINECALLFEATURE_UNHOLD
	For Park DNs:0
dwRemoveFromConfCaps	For All Devices:0
dwRemoveFromConfState	For All Devices:0
dwTransferModes	For IP Phones and CTI Ports:LINETRANSFERMODE_TRANSFER LINETRANSFERMODE_CONFERENCE
	For CTI Route Points and Park DNs:0
dwParkModes	For IP Phones and CTI Ports:LINEPARKMODE_NONDIRECTED
	For CTI Route Points and Park DNs:0
dwForwardModes	For All Devices (except ParkDNs):LINEFORWARDMODE_UNCOND
	For Park DNs:0
dwMaxForwardEntries	For All Devices (except ParkDNs):1
	For Park DNs:0

Members	Values
dwMaxSpecificEntries	For All Devices:0
dwMinFwdNumRings	For All Devices:0
dwMaxFwdNumRings	For All Devices:0
dwMaxCallCompletions	For All Devices:0
dwCallCompletionConds	For All Devices:0
dwCallCompletionModes	For All Devices:0
dwNumCompletionMessages	For All Devices:0
dwCompletionMsgTextEntrySize	For All Devices:0
dwCompletionMsgTextSize dwCompletionMsgTextOffset	For All Devices:0
dwAddressFeatures	For IP Phones and CTI Ports:LINEADDRFEATURE_FORWARD LINEADDRFEATURE_FORWARDFWD LINEADDRFEATURE_MAKECALL
	For CTI Route Points:LINEADDRFEATURE_FORWARD LINEADDRFEATURE_FORWARDFWD
	For Park DNs:0
dwPredictiveAutoTransferStates	For All Devices:0
dwNumCallTreatments	For All Devices:0
dwCallTreatmentListSizedwCallTreatmentListOffset	For All Devices:0
dwDeviceClassesSize dwDeviceClassesOffset	For All Devices (except Park DNs): "tapi/line" "tapi/phone" "wave/in" "wave/out"
	For Park DNs : "tapi/line"
dwMaxCallDataSize	For All Devices:0

Members	Values
dwCallFeatures2	For IP Phones and CTI Ports:LINECALLFEATURE2_TRANSFERNORM LINECALLFEATURE2_TRANSFERCONF
	For CTI Route Points and Park DNs:0
dwMaxNoAnswerTimeout	For IP Phones and CTI Ports: 4294967295 (0xFFFFFFFF)
	For CTI Route Points and Park DNs:0
dwConnectedModes	For IP Phones, CTI Ports LINECONNECTEDMODE_ACTIVE LINECONNECTEDMODE_INACTIVE
	For Park DNs: LINECONNECTEDMODE_ACTIVE
	For CTI Route Points (without media):0 For CTI Route Points (with media)LINECONNECTEDMODE_ACTIVE
dwOfferingModes	For All Devices: LINEOFFERINGMODE_ACTIVE
dwAvailableMediaModes	For All Devices:0

LINEADDRESSSTATUS

Members

Members	Values
dwNumInUse	For All Devices:1
dwNumActiveCalls	For All Devices: The number of calls on the address that are in call states other than idle, onhold, onholdpendingtransfer, and onholdpendingconference.
dwNumOnHoldCalls	For All Devices: The number of calls on the address in the onhold state.

Members	Values
dwNumOnHoldPendCalls	For All Devices: The number of calls on the address in the onholdpendingtransfer or the onholdpendingconference state.
dwAddressFeatures	For IP Phones and CTI Ports:LINEADDRFEATURE_FORWARD LINEADDRFEATURE_FORWARDFWD LINEADDRFEATURE_MAKECALL
	For CTI Route Points:LINEADDRFEATURE_FORWARD LINEADDRFEATURE_FORWARDFWD
	For Park DNs:0
dwNumRingsNoAnswer	For All Devices:0
dwForwardNumEntries	For All Devices (except Park DNs): The number of entries in the array to which dwForwardSize and dwForwardOffset refer.
	For Park DNs:0
dwForwardSize dwForwardOffset	For All Devices (except Park DNs): The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized field that describes the address forwarding information. This information appears as an array of dwForwardNumEntries elements, of type LINEFORWARD. Consider the offsets of the addresses in the array relative to the beginning of the LINEADDRESSSTATUS structure. The offsets dwCallerAddressOffset and dwDestAddressOffset in the variably sized field of type LINEFORWARD to which dwForwardSize and dwForwardOffset point are relative to the beginning of the LINEADDRESSSTATUS data structure (the root container).
	For Park DNs:0
dwTerminalModesSizedwTerminalModesOffset	For All Devices:0
dwDevSpecificSizedwDevSpecificOffset	For All Devices:0

LINEAPPINFO

The LINEAPPINFO structure contains information about the application that is currently running. The LINEDEVSTATUS structure can contain an array of LINEAPPINFO structures.

Structure Details

```
typedef struct lineappinfo_tag {
  DWORD  dwMachineNameSize;
  DWORD  dwMachineNameOffset;
  DWORD  dwUserNameSize;
  DWORD  dwUserNameOffset;
  DWORD  dwModuleFilenameSize;
  DWORD  dwModuleFilenameOffset;
  DWORD  dwFriendlyNameSize;
  DWORD  dwFriendlyNameOffset;
  DWORD  dwMediaModes;
  DWORD  dwAddressID;
} LINEAPPINFO, *LPLINEAPPINFO;
```

Members

Members	Values
dwMachineNameSize dwMachineNameOffset	Size (bytes) and offset from beginning of LINEDEVSTATUS of a string that specifies the name of the computer on which the application is executing.
dwUserNameSize dwUserNameOffset	Size (bytes) and offset from beginning of LINEDEVSTATUS of a string that specifies the user name under whose account the application is running.
dwModuleFilenameSize dwModuleFilenameOffset	Size (bytes) and offset from beginning of LINEDEVSTATUS of a string that specifies the module filename of the application. You can use this string in a call to lineHandoff to perform a directed handoff to the application.
dwFriendlyNameSize dwFriendlyNameOffset	Size (bytes) and offset from beginning of LINEDEVSTATUS of the string that the application provides to lineInitialize or lineInitializeEx, which should be used in any display of applications to the user.
dwMediaModes	The media types for which the application has requested ownership of new calls; zero if the line dwPrivileges did not include LINECALLPRIVILEGE_OWNER when it opened.
dwAddressID	If the line handle that was opened by using LINEOPENOPTION_SINGLEADDRESS contains the address identifier that is specified, set to 0xFFFFFFFF if the single address option was not used. An address identifier permanently associates with an address; the identifier remains constant across operating system upgrades.

LINECALLINFO

Members

Members	Values
hLine	For All Devices: The handle for the line device with which this call is associated.
dwLineDeviceID	For All Devices: The device identifier of the line device with which this call is associated.
dwAddressID	For All Devices:0
dwBearerMode	For All Devices: LINEBEARERMODE_SPEECH LINEBEARERMODE_VOICE
dwRate	For All Devices:0
dwMediaMode	For IP Phones and Park DNs:LINEMEDIAMODE_INTERACTIVEVOICE
	For CTI Ports and CTI Route Points:LINEMEDIAMODE_AUTOMATEDVOICE LINEMEDIAMODE_INTERACTIVEVOICE
dwAppSpecific	For All Devices: Not interpreted by the API implementation and service provider. Any owner application of this call can set it with the lineSetAppSpecific function.
dwCallID	For All Devices: In some telephony environments, the switch or service provider can assign a unique identifier to each call. This allows the call to be tracked across transfers, forwards, or other events. The domain of these call IDs and their scope is service provider-defined. The dwCallID member makes this unique identifier available to the applications. The Cisco Unified TSP uses dwCallID to store the “GlobalCallID” of the call. The “GlobalCallID” represents a unique identifier that allows applications to identify all call handles that are related to a call.
dwRelatedCallID	For All Devices:0
dwCallParamFlags	For All Devices:0

Members	Values
dwCallStates	For IP Phones and CTI Ports:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DIALING LINECALLSTATE_DIALTONE LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_ONHOLDPENDCONF LINECALLSTATE_ONHOLDPENDTRANSFER LINECALLSTATE_PROCEEDING LINECALLSTATE_RINGBACK LINECALLSTATE_UNKNOWN

Members	Values
dwCallStates (continued)	For CTI Route Points (without media):LINECALLSTATE_ACCEPTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_UNKNOWN For CTI Route Points (with media):LINECALLSTATE_ACCEPTED LINECALLSTATE_BUSY LINECALLSTATE_CONNECTED LINECALLSTATE_DIALING LINECALLSTATE_DIALTONE LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_PROCEEDING LINECALLSTATE_RINGBACK LINECALLSTATE_UNKNOWN
	For Park DNs:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_UNKNOWN
dwMonitorDigitModes	For IP Phones, CTI Ports, and CTI Route Points (with media): LINEDIGITMODE_DTMF
	For CTI Route Points and Park DNs:0

Members	Values
dwMonitorMediaModes	For IP Phones and Park DNs:LINEMEDIAMODE_INTERACTIVEVOICE
	For CTI Ports and CTI Route Points:LINEMEDIAMODE_AUTOMATEDVOICE LINEMEDIAMODE_INTERACTIVEVOICE
DialParams	For All Devices:0
dwOrigin	For All Devices:LINECALLORIGIN_CONFERENCE LINECALLORIGIN_EXTERNAL LINECALLORIGIN_INTERNAL LINECALLORIGIN_OUTBOUND LINECALLORIGIN_UNAVAIL LINECALLORIGIN_UNKNOWN
dwReason	For All Devices: LINECALLREASON_DIRECT LINECALLREASON_FWDBUSY LINECALLREASON_FWDNOANSWER LINECALLREASON_FWDUNCOND LINECALLREASON_PARKED LINECALLREASON_PICKUP LINECALLREASON_REDIRECT LINECALLREASON_REMINDER LINECALLREASON_TRANSFER LINECALLREASON_UNKNOWN LINECALLREASON_UNPARK
dwCompletionID	For All Devices:0
dwNumOwners	For All Devices: The number of application modules with different call handles with owner privilege for the call.
dwNumMonitors	For All Devices: The number of application modules with different call handles with monitor privilege for the call.
dwCountryCode	For All Devices:0
dwTrunk	For All Devices:0xFFFFFFFF

Members	Values
dwCallerIDFlags	For All Devices:LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED
dwCallerIDSize dwCallerIDOffset	For All Devices: The size, in bytes, of the variably sized field that contains the caller party ID number information and the offset, in bytes, from the beginning of this data structure.
dwCallerIDNameSize dwCallerIDNameOffset	For All Devices: The size, in bytes, of the variably sized field that contains the caller party ID name information and the offset, in bytes, from the beginning of this data structure.
dwCalledIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN
dwCalledIDSize dwCalledIDOffset	For All Devices: The size, in bytes, of the variably sized field that contains the called-party ID number information and the offset, in bytes, from the beginning of this data structure.
dwCalledIDNameSize dwCalledIDNameOffset	For All Devices: The size, in bytes, of the variably sized field that contains the called-party ID name information and the offset, in bytes, from the beginning of this data structure.
dwConnectedIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED
dwConnectedIDSize dwConnectedIDOffset	For All Devices: The size, in bytes, of the variably sized field that contains the connected party identifier number information and the offset, in bytes, from the beginning of this data structure.

Members	Values
dwConnectedIDNameSize dwConnectedIDNameOffset	For All Devices: The size, in bytes, of the variably sized field that contains the connected party identifier name information and the offset, in bytes, from the beginning of this data structure.
dwRedirectionIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_BLOCKED
dwRedirectionIDSize dwRedirectionIDOffset	For All Devices: The size, in bytes, of the variably sized field that contains the redirection party identifier number information and the offset, in bytes, from the beginning of this data structure.
dwRedirectionIDNameSize dwRedirectionIDNameOffset	For All Devices: The size, in bytes, of the variably sized field that contains the redirection party identifier name information and the offset, in bytes, from the beginning of this data structure.
dwRedirectingIDFlags	For All Devices: LINECALLPARTYID_ADDRESS LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN
dwRedirectingIDSize dwRedirectingIDOffset	For All Devices: The size, in bytes, of the variably sized field that contains the redirecting party identifier number information and the offset, in bytes, from the beginning of this data structure.
dwRedirectingIDNameSize dwRedirectingIDNameOffset	For All Devices: The size, in bytes, of the variably sized field that contains the redirecting party identifier name information and the offset, in bytes, from the beginning of this data structure.
dwAppNameSize dwAppNameOffset	For All Devices: The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized field that holds the user-friendly application name of the application that first originated, accepted, or answered the call. This specifies the name that an application can specify in lineInitializeEx. If the application specifies no such name, the application module filename gets used instead.

Members	Values
dwDisplayableAddressSize dwDisplayableAddressOffset	For All Devices: 0
dwCalledPartySize dwCalledPartyOffset	For All Devices: 0
dwCommentSize dwCommentOffset	For All Devices: 0
dwDisplaySize dwDisplayOffset	For All Devices: 0
dwUserUserInfoSize dwUserUserInfoOffset	For All Devices: 0
dwHighLevelCompSize dwHighLevelCompOffset	For All Devices: 0
dwLowLevelCompSize dwLowLevelCompOffset	For All Devices: 0
dwChargingInfoSize dwChargingInfoOffset	For All Devices: 0
dwTerminalModesSize dwTerminalModesOffset	For All Devices: 0

Members	Values
dwDevSpecificSize dwDevSpecificOffset	For All Devices: If dwExtVersion > = 0x00060000 (6.0), this field will point to TSP_Unicode_Party_Names structure, If dwExtVersion > = 0x00070000 (7.0), this field will also point to a common structure that has a pointer to SRTP structure, DSCPValueForAudioCalls value, and Partition information. The LINECALLINFO, on page 320 defines the structure. The ExtendedCallInfo structure contains ExtendedCallReason that represents the last feature-related reason that caused a change in the callinfo/callstatus for this call. The ExtendedCallInfo will also provide SIP URL information for all call parties. If dwExtVersion > = 0x00080000 (8.0), this field will also point to common structure which has pointer to CallSecurityStatus structure. For IP Phones: If dwExtVersion > = 0x00080000 (8.0), this field will also point to common structure that has pointer to CallAtributeInfo and CCMCallID structure. The structures are defined below. If dwExtVersion > = 0x00080000 (8.0), this field will also point to common structure which has pointer to CallSecurityStatus structure.
	CallAttributeType: This field holds information about DN.Partition.DeviceName for regular calls, monitoring calls, monitored calls, and recording calls. PartyDNOffset, PartyDNSize, provides the size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party DN information and the offset, in bytes, from the beginning of LINECALLINFO data structure. PartyPartitionOffset PartyPartitionSize, provides the size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party Partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure. DevcieNameSizeprovides the size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party Device Name and the offset, in bytes, from the beginning of LINECALLINFO data structure. OverallCallSecurityStatus holds the security status of the call for two-party call as well for conference call. CCMCallID field holds the CCM call Id for each call leg.
dwCallTreatment	For All Devices: 0

Members	Values
dwCallDataSize dwCallDataOffset	For All Devices: 0
dwSendingFlowspecSize dwSendingFlowspecOffset	For All Devices: 0
dwReceivingFlowspecSize dwReceivingFlowspecOffset	For All Devices: 0

LINECALLLIST

The LINECALLLIST structure describes a list of call handles. The lineGetNewCalls and lineGetConfRelatedCalls functions return a structure of this type.

Note You must not extend this structure.

Structure Details

```
typedef struct linecalllist_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwCallsNumEntries;
  DWORD  dwCallsSize;
  DWORD  dwCallsOffset;
} LINECALLLIST, FAR *LPLINECALLLIST;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that is allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.
dwCallsNumEntries	The number of handles in the hCalls array.
dwCallsSized wCallsOffset	The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized field (which is an array of HCALL-sized handles).

LINECALLPARAMS

Members

Members	Values
dwBearerMode	not supported
dwMinRatedwMaxRate	not supported
dwMediaMode	not supported
dwCallParamFlags	not supported
dwAddressMode	not supported
dwAddressID	not supported
DialParams	not supported
dwOrigAddressSize dwOrigAddressOffset	not supported
dwDisplayableAddressSize dwDisplayableAddressOffset	not supported
dwCalledPartySize dwCalledPartyOffset	not supported
dwCommentSize dwCommentOffset	not supported
dwUserUserInfoSize dwUserUserInfoOffset	not supported
dwHighLevelCompSize dwHighLevelCompOffset	not supported
dwLowLevelCompSize dwLowLevelCompOffset	not supported
dwDevSpecificSize dwDevSpecificOffset	not supported
dwPredictiveAutoTransferStates	not supported
dwTargetAddressSize dwTargetAddressOffset	not supported

Members	Values
dwSendingFlowspecSize dwSendingFlowspecOffset	not supported
dwReceivingFlowspecSize dwReceivingFlowspecOffset	not supported
dwDeviceClassSize dwDeviceClassOffset	not supported
dwDeviceConfigSize dwDeviceConfigOffset	not supported
dwCallDataSize dwCallDataOffset	not supported
dwNoAnswerTimeout	For All Devices: The number of seconds, after the completion of dialing, that the call should be allowed to wait in the PROCEEDING or RINGBACK state before the service provider automatically abandons it with a LINECALLSTATE_DISCONNECTED and LINEDISCONNECTMODE_NOANSWER. A value of 0 indicates that the application does not want automatic call abandonment.
dwCallingPartyIDSize dwCallingPartyIDOffset	not supported

LINECALLSTATUS

Members

Members	Values
dwCallState	For IP Phones and CTI Ports:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DIALING LINECALLSTATE_DIALTONE LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_ONHOLDPENDCONF LINECALLSTATE_ONHOLDPENDTRANSFER LINECALLSTATE_PROCEEDING LINECALLSTATE_RINGBACK LINECALLSTATE_UNKNOWN

Members	Values
	For CTI Route Points (without media):LINECALLSTATE_ACCEPTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_UNKNOWN For CTI Route Points (with media):LINECALLSTATE_ACCEPTED LINECALLSTATE_CONNECTED LINECALLSTATE_DIALING LINECALLSTATE_DIALTONE LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_PROCEEDING LINECALLSTATE_RINGBACK LINECALLSTATE_UNKNOWN
dwCallState (continued)	For Park DNs:LINECALLSTATE_ACCEPTED LINECALLSTATE_CONFERENCED LINECALLSTATE_CONNECTED LINECALLSTATE_DISCONNECTED LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_ONHOLD LINECALLSTATE_UNKNOWN

Members	Values
dwCallStateMode	For IP Phones, CTI Ports:LINECONNECTEDMODE_ACTIVE LINECONNECTEDMODE_INACTIVE LINEDIALTONEMODE_NORMAL LINEDIALTONEMODE_UNAVAIL LINEDISCONNECTMODE_BADADDRESS LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_TEMPFAILURE LINEDISCONNECTMODE_UNREACHABLE LINEDISCONNECTMODE_FACCMC (if negotiated extension version is 0x00050000 or greater)
	For CTI Route Points:LINEDISCONNECTMODE_BADADDRESS LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_TEMPFAILURE LINEDISCONNECTMODE_UNREACHABLE LINEDISCONNECTMODE_FACCMC (if negotiated extension version is 0x00050000 or greater)

Members	Values
	For Park DNs:LINECONNECTEDMODE_ACTIVE LINEDISCONNECTMODE_BADADDRESS LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_TEMPFAILURE LINEDISCONNECTMODE_UNREACHABLE
dwCallPrivilege	For All Devices LINECALLPRIVILEGE_MONITOR LINECALLPRIVILEGE_NONE LINECALLPRIVILEGE_OWNER

Members	Values
dwCallFeatures	For IP Phones (except VG248 and ATA186) and CTI Ports: LINECALLFEATURE_ACCEPT LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_ANSWER LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_PARK LINECALLFEATURE_PREPAREADDTOCONF LINECALLFEATURE_REDIRECT LINECALLFEATURE_SETUPCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_UNHOLD LINECALLFEATURE_UNPARK

Members	Values
	For VG248 and ATA186 Devices:LINECALLFEATURE_ACCEPT LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_PARK LINECALLFEATURE_PREPAREADDTOCONF LINECALLFEATURE_REDIRECT LINECALLFEATURE_SETUPCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_UNHOLD LINECALLFEATURE_UNPARK

Members	Values
dwCallFeatures (continued)	For CTI Route Points (without media):LINECALLFEATURE_ACCEPT LINECALLFEATURE_DROP LINECALLFEATURE_REDIRECT For CTI Route Points (with media):LINECALLFEATURE_ACCEPT LINECALLFEATURE_ANSWER LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_DIA LLINECALLFEATURE_DROP LINECALLFEATURE_GATHERDIGITS LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_GENERATETONE LINECALLFEATURE_HOLD LINECALLFEATURE_MONITORDIGITS LINECALLFEATURE_MONITORTONES LINECALLFEATURE_REDIRECT LINECALLFEATURE_UNHOLD
dwCallFeatures (continued)	For Park DNs:0
dwDevSpecificSizedwDevSpecificOffset	For All Devices:0
dwCallFeatures2	For IP Phones and CTI Ports:LINECALLFEATURE2_TRANSFERNORM LINECALLFEATURE2_TRANSFERCONF
	For CTI Route Points and Park DNs:0
tStateEntryTime	For All Devices: The Coordinated Universal Time at which the current call state was entered.

LINECARDENTRY

The LINECARDENTRY structure describes a calling card. The LINETRANSLATECAPS structure can contain an array of LINECARDENTRY structures.

Note You must not extend this structure.

Structure Details

```
typedef struct linecardentry_tag {
  DWORD  dwPermanentCardID;
  DWORD  dwCardNameSize;
  DWORD  dwCardNameOffset;
  DWORD  dwCardNumberDigits;
  DWORD  dwSameAreaRuleSize;
  DWORD  dwSameAreaRuleOffset;
  DWORD  dwLongDistanceRuleSize;
  DWORD  dwLongDistanceRuleOffset;
  DWORD  dwInternationalRuleSize;
  DWORD  dwInternationalRuleOffset;
  DWORD  dwOptions;
} LINECARDENTRY, FAR *LPLINECARDENTRY;
```

Members

Members	Values
dwPermanentCardID	The permanent identifier that identifies the card.
dwCardNameSize dwCardNameOffset	A null-terminated string (size includes the NULL) that describes the card in a user-friendly manner.
dwCardNumberDigits	The number of digits in the existing card number. The card number itself is not returned for security reasons (TAPI stores it in scrambled form). The application can use this parameter to insert filler bytes into a text control in “password” mode to show that a number exists.
dwSameAreaRuleSize dwSameAreaRuleOffset	The offset, in bytes, from the beginning of the LINETRANSLATECAPS structure and the total number of bytes in the dialing rule that is defined for calls to numbers in the same area code. The rule specifies a null-terminated string.
dwLongDistanceRuleSize dwLongDistanceRuleOffset	The offset, in bytes, from the beginning of the LINETRANSLATECAPS structure and the total number of bytes in the dialing rule that is defined for calls to numbers in the other areas in the same country or region. The rule specifies a null-terminated string.
dwInternationalRuleSize dwInternationalRuleOffset	The offset, in bytes, from the beginning of the LINETRANSLATECAPS structure and the total number of bytes in the dialing rule that is defined for calls to numbers in other countries/regions. The rule specifies a null-terminated string.
dwOptions	Indicates other settings that are associated with this calling card, by using the LINECARDOPTION_

LINECOUNTRYENTRY

The LINECOUNTRYENTRY structure provides the information for a single country entry. An array of one or more of these structures makes up part of the LINECOUNTRYLIST structure that the lineGetCountry function returns.

Note You must not extend this structure.

Structure Details

```
typedef struct linecountryentry_tag {
  DWORD  dwCountryID;
  DWORD  dwCountryCode;
  DWORD  dwNextCountryID;
  DWORD  dwCountryNameSize;
  DWORD  dwCountryNameOffset;
  DWORD  dwSameAreaRuleSize;
  DWORD  dwSameAreaRuleOffset;
  DWORD  dwLongDistanceRuleSize;
  DWORD  dwLongDistanceRuleOffset;
  DWORD  dwInternationalRuleSize;
  DWORD  dwInternationalRuleOffset;
} LINECOUNTRYENTRY, FAR *LPLINECOUNTRYENTRY;
```

Members

Members	Values
dwCountryID	The country or region identifier of the entry that specifies an internal identifier that allows multiple entries to exist in the country or region list with the same country code (for example, all countries in North America and the Caribbean share country code 1, but require separate entries in the list).
dwCountryCode	The actual country code of the country or region that the entry represents (that is, the digits that would be dialed in an international call). Display only this value to users (Country IDs should never display, as they could be confusing).
dwNextCountryID	The country identifier of the next entry in the country or region list. Because country codes and identifiers are not assigned in numeric sequence, the country or region list represents a single linked list, with each entry pointing to the next. The last country or region in the list includes a dwNextCountryID value of zero. When the LINECOUNTRYLIST structure is used to obtain the entire list, the entries in the list appear in sequence as linked by their dwNextCountryID members.
dwCountryNameSize dwCountryNameOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINECOUNTRYLIST structure of a null-terminated string that gives the name of the country or region.

Members	Values
dwSameAreaRuleSize dwSameAreaRuleOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINECOUNTRYLIST structure of a null-terminated string that contains the dialing rule for direct-dialed calls to the same area code.
dwLongDistanceRuleSize dwLongDistanceRuleOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINECOUNTRYLIST structure of a null-terminated string that contains the dialing rule for direct-dialed calls to other areas in the same country or region.
dwInternationalRuleSize dwInternationalRuleOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINECOUNTRYLIST structure of a null-terminated string that contains the dialing rule for direct-dialed calls to other countries/regions.

LINECOUNTRYLIST

The LINECOUNTRYLIST structure describes a list of countries/regions. This structure can contain an array of LINECOUNTRYENTRY structures. The lineGetCountry function returns LINECOUNTRYLIST.

Note You must not extend this structure.

Structure Details

```
typedef struct linecountrylist_tag {
 DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwNumCountries;
  DWORD  dwCountryListSize;
  DWORD  dwCountryListOffset;
} LINECOUNTRYLIST, FAR *LPLINECOUNTRYLIST;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that are allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.

Members	Values
dwNumCountries	The number of LINECOUNTRYENTRY structures that are present in the array dwCountryListSize and dwCountryListOffset dominate.
dwCountryListSize dwCountryListOffset	The size, in bytes, and the offset, in bytes, from the beginning of this data structure of an array of LINECOUNTRYENTRY elements that provide information on each country or region.

LINEDEVCAPS

Members

Members	Values
dwProviderInfoSize dwProviderInfoOffset	For All Devices: The size, in bytes, of the variably sized field that contains service provider information and the offset, in bytes, from the beginning of this data structure. The dwProviderInfoSize/ Offset member provides information about the provider hardware and/or software. This information is useful when a user needs to call customer service with problems regarding the provider. The Cisco Unified TSP sets this field to "Cisco Unified TSPxxx.TSP: Cisco IP PBX Service Provider Ver. x.x(x.x)" where the text before the colon specifies the file name of the TSP and the text after "Ver." specifies the version of TSP.
dwSwitchInfoSize dwSwitchInfoOffset	For All Devices: The size, in bytes, of the variably sized device field that contains switch information and the offset, in bytes, from the beginning of this data structure. The dwSwitchInfoSize/Offset member provides information about the switch to which the line device connects, such as the switch manufacturer, the model name, the software version, and so on. This information is useful when a user needs to call customer service with problems regarding the switch. The Cisco Unified TSP sets this field to "Cisco Unified Communications Manager Ver. x.x(x.x), Cisco CTI Manager Ver x.x(x.x)" where the text after "Ver." specifies the version of the Cisco Unified Communications Manager and the version of the CTI Manager, respectively.

Members	Values
dwPermanentLineID	For All Devices: The permanent DWORD identifier by which the line device is known in the system configuration. This identifier specifies a permanent name for the line device. This permanent name (as opposed to dwDeviceID) does not change as lines are added or removed from the system and persists through operating system upgrades. You can therefore use it to link line-specific information in .ini files (or other files) in a way that is not affected by adding or removing other lines or by changing the operating system.
dwLineNameSize dwLineNameOffset	For All Devices: The size, in bytes, of the variably sized device field that contains a user-configurable name for this line device and the offset, in bytes, from the beginning of this data structure. You can configure this name when you configure the line device service provider, and the name gets provided for the convenience of the user. Cisco Unified TSP sets this field to “Cisco Line: [deviceName] (dirn)” where deviceName specifies the name of the device on which the line resides, and dirn specifies the directory number for the device.
dwStringFormat	For All Devices: STRINGFORMAT_ASCII
dwAddressModes	For All Devices: LINEADDRESSMODE_ADDRESSID
dwNumAddresses	For All Devices:1
dwBearerModes	For All Devices: LINEBEARERMODE_SPEECH LINEBEARERMODE_VOICE
dwMaxRate	For All Devices:0
dwMediaModes	For IP Phones and Park DNs: LINEMEDIAMODE_INTERACTIVEVOICE
	For CTI Ports and CTI Route Points: LINEMEDIAMODE_AUTOMATEDVOICE LINEMEDIAMODE_INTERACTIVEVOICE
dwGenerateToneModes	For IP Phones, CTI Ports, and CTI Route Points (with media): LINETONEMODE_BEEP
	For CTI Route Points (without media) and Park DNs:0
dwGenerateToneMaxNumFreq	For All Devices:0

Members	Values
dwGenerateDigitModes	For IP Phones, CTI Ports, and CTI Route Points (with media): LINETONEMODE_DTMF
	For CTI Route Points and Park DNs:0
dwMonitorToneMaxNumFreq	For All Devices:0
dwMonitorToneMaxNumEntries	For All Devices:0
dwMonitorDigitModes	For IP Phones, CTI Ports, and CTI Route Points (with media): LINETONEMODE_DTMF
	For CTI Route Points (without media) and Park DNs:0
dwGatherDigitsMinTimeout dwGatherDigitsMaxTimeout	For All Devices:0
dwMedCtlDigitMaxListSize dwMedCtlMediaMaxListSize dwMedCtlToneMaxListSize dwMedCtlCallStateMaxListSize	For All Devices:0
dwDevCapFlags	For IP Phones:0
	For All Other Devices: LINEDEVCAPFLAGS_CLOSEDROP
dwMaxNumActiveCalls	For All Devices:1
	For CTI Route Points (without media):0 For CTI Route Points (with media): Cisco Unified Communications Manager Administration configuration
dwAnswerMode	For IP Phones (except for VG248 and ATA186), CTI Route Points (with media) and CTI Ports: LINEANSWERMODE_HOLD
	For VG248 devices, ATA186 devices, CTI Route Points (without media), and Park DNs:0
dwRingModes	For All Devices:1

Members	Values
dwLineStates	For IP Phones, CTI Ports, and Route Points (with media): LINEDEVSTATE_CLOSE LINEDEVSTATE_DEVSPECIFIC LINEDEVSTATE_INSERVICE LINEDEVSTATE_MSGWAITOFF LINEDEVSTATE_MSGWAITON LINEDEVSTATE_NUMCALLS LINEDEVSTATE_OPEN LINEDEVSTATE_OUTOFSERVICE LINEDEVSTATE_REINIT LINEDEVSTATE_RINGING LINEDEVSTATE_TRANSLATECHANGE
	For CTI Route Points (without media): LINEDEVSTATE_CLOSE LINEDEVSTATE_INSERVICE LINEDEVSTATE_OPEN LINEDEVSTATE_OUTOFSERVICE LINEDEVSTATE_REINIT LINEDEVSTATE_RINGING LINEDEVSTATE_TRANSLATECHANGE
	For Park DNs:LINEDEVSTATE_CLOSE LINEDEVSTATE_DEVSPECIFIC LINEDEVSTATE_INSERVICE LINEDEVSTATE_NUMCALLS LINEDEVSTATE_OPEN LINEDEVSTATE_OUTOFSERVICE LINEDEVSTATE_REINIT LINEDEVSTATE_TRANSLATECHANGE
dwUUIAcceptSize	For All Devices:0
dwUUIAnswerSize	For All Devices:0
dwUUIMakeCallSize	For All Devices:0
dwUUIDropSize	For All Devices:0

Members	Values
dwUUISendUserUserInfoSize	For All Devices:0
dwUUICallInfoSize	For All Devices:0
MinDialParams MaxDialParams	For All Devices:0
DefaultDialParams	For All Devices:0
dwNumTerminals	For All Devices:0
dwTerminalCapsSize dwTerminalCapsOffset	For All Devices:0
dwTerminalTextEntrySize	For All Devices:0
dwTerminalTextSize dwTerminalTextOffset	For All Devices:0
dwDevSpecificSize dwDevSpecificOffset	For All Devices (except ParkDNs): If dwExtVersion > 0x00030000 (3.0):LINEDEVCAPS_DEV_SPECIFIC.m_DevSpecificFlags = 0
	For Park DNs: If dwExtVersion > 0x00030000 (3.0):LINEDEVCAPS_DEV_SPECIFIC.m_DevSpecificFlags = LINEDEVCAPSDEVSPECIFIC_PARKDN For Intercom DNs: LINEDEVCAPS_DEV_SPECIFIC. M_DevSpecificFlags = LINEDEVCAPSDEVSPECIFIC_INTERCOMDNLOCALE info PARTITION_INFO INTERCOM_SPEEDDIAL_INFO
dwLineFeatures	For IP Phones, CTI Ports, and CTI Route Points (with media): LINEFEATURE_DEVSPECIFIC LINEFEATURE_FORWARD LINEFEATURE_FORWARDFWD LINEFEATURE_MAKECALL
	For CTI Route Points (without media): LINEFEATURE_FORWARD LINEFEATURE_FORWARDFWD
	For Park DNs:0

Members	Values
dwSettableDevStatus	For All Devices:0
dwDeviceClassesSize dwDeviceClassesOffset	For IP Phones and CTI Route Points: "tapi/line" "tapi/phone"
	For CTI Ports: "tapi/line" "tapi/phone" "wave/in" "wave/out"
	For Park DNs: "tapi/line"
PermanentLineGuid	The GUID that is permanently associated with the line device.

LINEDEVSTATUS

Members

Members	Values
dwNumOpens	For All Devices: The number of active opens on the line device.
dwOpenMediaModes	For All Devices: Bit array that indicates for which media types the line device is currently open.
dwNumActiveCalls	For All Devices: The number of calls on the line in call states other than idle, onhold, onholdpendingtransfer, and onholdpendingconference.
dwNumOnHoldCalls	For All Devices: The number of calls on the line in the onhold state.
dwNumOnHoldPendCalls	For All Devices: The number of calls on the line in the onholdpendingtransfer or onholdpendingconference state.

Members	Values
dwLineFeatures	For IP Phones, CTI Ports, and CTI Route Points (with media): LINEFEATURE_DEVSPECIFIC LINEFEATURE_FORWARD LINEFEATURE_FORWARDFWD LINEFEATURE_MAKECALL
	For CTI Route Points (without media): LINEFEATURE_FORWARD LINEFEATURE_FORWARDFWD
	For Park DNs:0
dwNumCallCompletions	For All Devices:0
dwRingMode	For All Devices:0
dwSignalLevel	For All Devices:0
dwBatteryLevel	For All Devices:0
dwRoamMode	For All Devices:0
dwDevStatusFlags	For IP Phones and CTI Ports: LINEDEVSTATUSGLAGS_CONNECTED LINEDEVSTATUSGLAGS_INSERVICE LINEDEVSTATUSGLAGS_MSGWAIT
	For CTI Route Points and Park DNs: LINEDEVSTATUSGLAGS_CONNECTED LINEDEVSTATUSGLAGS_INSERVICE
dwTerminalModesSizedwTerminalModesOffset	For All Devices:0
dwDevSpecificSizedwDevSpecificOffset	For All Devices:0
dwAvailableMediaModes	For All Devices:0
dwAppInfoSizedwAppInfoOffset	For All Devices: Length, in bytes, and offset from the beginning of LINEDEVSTATUS of an array of LINEAPPINFO structures. The dwNumOpens member indicates the number of elements in the array. Each element in the array identifies an application that has the line open.

LINEEXTENSIONID

Members

Members	Values
dwExtensionID0	For All Devices: 0x8EBD6A50
dwExtensionID1	For All Devices: 0x128011D2
dwExtensionID2	For All Devices: 0x905B0060
dwExtensionID3	For All Devices: 0xB03DD275

LINEFORWARD

The LINEFORWARD structure describes an entry of the forwarding instructions.

Structure Details

```
typedef struct lineforward_tag {
    DWORD  dwForwardMode;
    DWORD  dwCallerAddressSize;
    DWORD  dwCallerAddressOffset;
    DWORD  dwDestCountryCode;
    DWORD  dwDestAddressSize;
    DWORD  dwDestAddressOffset;
} LINEFORWARD, FAR *LPLINEFORWARD;
```

Members

Members	Values
dwForwardMode	

Members	Values
	The types of forwarding. The dwForwardMode member can have only a single bit set. This member uses the following LINEFORWARDMODE_ constants: **LINEFORWARDMODE_UNCOND** Forward all calls unconditionally, irrespective of their origin. Use this value when unconditional forwarding for internal and external calls cannot be controlled separately. Unconditional forwarding overrides forwarding on busy and/or no-answer conditions. **Note** LINEFORWARDMODE_UNCOND is the only forward mode that Cisco Unified TSP supports. **LINEFORWARDMODE_UNCONDINTERNAL** Forward all internal calls unconditionally. Use this value when unconditional forwarding for internal and external calls can be controlled separately. **LINEFORWARDMODE_UNCONDEXTERNAL** Forward all external calls unconditionally. Use this value when unconditional forwarding for internal and external calls can be controlled separately. **LINEFORWARDMODE_UNCONDSPECIFIC** Unconditionally forward all calls that originated at a specified address (selective call forwarding). **LINEFORWARDMODE_BUSY** Forward all calls on busy, irrespective of their origin. Use this value when forwarding for internal and external calls both on busy and on no answer cannot be controlled separately. **LINEFORWARDMODE_BUSYINTERNAL** Forward all internal calls on busy. Use this value when forwarding for internal and external calls on busy and on no answer can be controlled separately. **LINEFORWARDMODE_BUSYEXTERNAL** Forward all external calls on busy. Use this value when forwarding for internal and external calls on busy and on no answer can be controlled separately. **LINEFORWARDMODE_BUSYSPECIFIC** Forward on busy all calls that originated at a specified address (selective call forwarding). **LINEFORWARDMODE_NOANSW**

Members	Values
	Forward all calls on no answer, irrespective of their origin. Use this value when call forwarding for internal and external calls on no answer cannot be controlled separately. **LINEFORWARDMODE_NOANSWINTERNAL** Forward all internal calls on no answer. Use this value when forwarding for internal and external calls on no answer can be controlled separately. **LINEFORWARDMODE_NOANSWEXTERNAL** Forward all external calls on no answer. Use this value when forwarding for internal and external calls on no answer can be controlled separately. **LINEFORWARDMODE_NOANSWSPECIFIC** Forward all calls that originated at a specified address on no answer (selective call forwarding). **LINEFORWARDMODE_BUSYNA** Forward all calls on busy or no answer, irrespective of their origin. Use this value when forwarding for internal and external calls on both busy and on no answer cannot be controlled separately. **LINEFORWARDMODE_BUSYNAINTERNAL** Forward all internal calls on busy or no answer. Use this value when call forwarding on busy and on no answer cannot be controlled separately for internal calls. **LINEFORWARDMODE_BUSYNAEXTERNAL** Forward all external calls on busy or no answer. Use this value when call forwarding on busy and on no answer cannot be controlled separately for internal calls. **LINEFORWARDMODE_BUSYNASPECIFIC** Forward on busy or no answer all calls that originated at a specified address (selective call forwarding). **LINEFORWARDMODE_UNKNOWN** Calls get forwarded, but the conditions under which forwarding occurs are not known at this time. **LINEFORWARDMODE_UNAVAIL** Calls are forwarded, but the conditions under which forwarding occurs are not known and are never known by the service provider.

Members	Values
dwCallerAddressSize dwCallerAddressOffset	The size in bytes of the variably sized address field that contains the address of a caller to be forwarded and the offset in bytes from the beginning of the containing data structure. The dwCallerAddressSize/Offset member gets set to zero if dwForwardMode is not one of the following choices:LINEFORWARDMODE_BUSYNASPECIFIC, LINEFORWARDMODE_NOANSWSPECIFIC, LINEFORWARDMODE_UNCONDSPECIFIC, or LINEFORWARDMODE_BUSYSPECIFIC.
dwDestCountryCode	The country code of the destination address to which the call is to be forwarded.
dwDestAddressSize dwDestAddressOffset	The size in bytes of the variably sized address field that contains the address where calls are to be forwarded and the offset in bytes from the beginning of the containing data structure.

LINEFORWARDLIST

The LINEFORWARDLIST structure describes a list of forwarding instructions.

Structure Details

```
typedef struct lineforwardlist_tag {
    DWORD  dwTotalSize;
    DWORD  dwNumEntries;
    LINEFORWARD  ForwardList[1];
} LINEFORWARDLIST, FAR *LPLINEFORWARDLIST;
```

Members

Members	Values
dwTotalSize	The total size in bytes of the data structure.
dwNumEntries	Number of entries in the array, specified as ForwardList[ ].
ForwardList[ ]	An array of forwarding instruction. The array entries specify type LINEFORWARD.

LINEGENERATETONE

The LINEGENERATETONE structure contains information about a tone to be generated. The lineGenerateTone and TSPI_lineGenerateTone functions use this structure.

Note You must not extend this structure.

This structure gets used only for the generation of tones; it is not used for tone monitoring.

Structure Details

```
typedef struct linegeneratetone_tag {
  DWORD  dwFrequency;
  DWORD  dwCadenceOn;
  DWORD  dwCadenceOff;
  DWORD  dwVolume;
} LINEGENERATETONE, FAR *LPLINEGENERATETONE;
```

Members

Members	Values
dwFrequency	The frequency, in hertz, of this tone component. A service provider may adjust (round up or down) the frequency that the application specified to fit its resolution.
dwCadenceOn	The “on” duration, in milliseconds, of the cadence of the custom tone to be generated. Zero means no tone gets generated.
dwCadenceOff	The “off” duration, in milliseconds, of the cadence of the custom tone to be generated. Zero means no off time, that is, a constant tone.
dwVolume	The volume level at which the tone gets generated. A value of 0x0000FFFF represents full volume, and a value of 0x00000000 means silence.

LINEINITIALIZEEXPARAMS

The LINEINITIZALIZEEXPARAMS structure describes parameters that are supplied when calls are made by using LINEINITIALIZEEX.

Structure Details

```
typedef struct lineinitializeexparams_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwOptions;

 union
 {
  HANDLE  hEvent;
  HANDLE  hCompletionPort;
 } Handles;
```

```
  DWORD  dwCompletionKey;
} LINEINITIALIZEEXPARAMS, FAR *LPLINEINITIALIZEEXPARAMS;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that is allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.
dwOptions	One of the LINEINITIALIZEEXOPTION_constants. Specifies the event notification mechanism that the application wants to use.
hEvent	If dwOptions specifies LINEINITIALIZEEXOPTION_USEEVENT, TAPI returns the event handle in this field.
hCompletionPort	If dwOptions specifies LINEINITIALIZEEXOPTION_USECOMPLETIONPORT, the application must specify in this field the handle of an existing completion port that was opened by using CreateIoCompletionPort.
dwCompletionKey	If dwOptions specifies LINEINITIALIZEEXOPTION_USECOMPLETIONPORT, the application must specify in this field a value that is returned through the lpCompletionKey parameter of GetQueuedCompletionStatus to identify the completion message as a telephony message.

Further Details

See lineInitializeEx, on page 172 for further information on these options.

LINELOCATIONENTRY

The LINELOCATIONENTRY structure describes a location that is used to provide an address translation context. The LINETRANSLATECAPS structure can contain an array of LINELOCATIONENTRY structures.

Note You must not extend this structure.

Structure Details

```
typedef struct linelocationentry_tag {
  DWORD  dwPermanentLocationID;
  DWORD  dwLocationNameSize;
  DWORD  dwLocationNameOffset;

  DWORD  dwCityCodeSize;
  DWORD  dwCityCodeOffset;
  DWORD  dwPreferredCardID;
  DWORD  dwLocalAccessCodeSize;
  DWORD  dwLocalAccessCodeOffset;
  DWORD  dwLongDistanceAccessCodeSize;
  DWORD  dwLongDistanceAccessCodeOffset;
  DWORD  dwTollPrefixListSize;
  DWORD  dwTollPrefixListOffset;
  DWORD  dwCountryID;
  DWORD  dwOptions;
  DWORD  dwCancelCallWaitingSize;
  DWORD  dwCancelCallWaitingOffset;
} LINELOCATIONENTRY, FAR *LPLINELOCATIONENTRY;
```

Members

Members	Values
dwPermanentLocationID	The permanent identifier that identifies the location.
dwLocationNameSize dwLocationNameOffset	Contains a null-terminated string (size includes the NULL) that describes the location in a user-friendly manner.
dwCountryCode	The country code of the location.
dwPreferredCardID	The preferred calling card when dialing from this location.
dwCityCodeSize dwCityCodeOffset	Contains a null-terminated string that specifies the city or area code that is associated with the location (the size includes the NULL). Applications can use this information, along with the country code, to “default” entry fields for the user when you enter the phone numbers, to encourage the entry of proper canonical numbers.
dwLocalAccessCodeSize dwLocalAccessCodeOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINETRANSLATECAPS structure of a null-terminated string that contains the access code to be dialed before calls to addresses in the local calling area.
dwLongDistanceAccessCodeSize dwLongDistanceAccessCodeOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINETRANSLATECAPS structure of a null-terminated string that contains the access code to be dialed before calls to addresses outside the local calling area.

Members	Values
dwTollPrefixListSize dwTollPrefixListOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINETRANSLATECAPS structure of a null-terminated string that contains the toll prefix list for the location. The string contains only prefixes that consist of the digits “0” through “9” and are separated from each other by a single “,” (comma) character.
dwCountryID	The country identifier of the country or region that is selected for the location. Use this identifier with the lineGetCountry function to obtain additional information about the specific country or region, such as the country or region name (you cannot use the dwCountryCode member for this purpose because country codes are not unique).
dwOptions	Indicates options in effect for this location with values taken from the LINELOCATIONOPTION_Constants.
dwCancelCallWaitingSize dwCancelCallWaitingOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINETRANSLATECAPS structure of a null-terminated string that contains the dial digits and modifier characters that should be prefixed to the dialable string (after the pulse/tone character) when an application sets the LINETRANSLATEOPTION_CANCELCALLWAITING bit in the dwTranslateOptions parameter of lineTranslateAddress. If no prefix is defined, dwCancelCallWaitingSize set to zero may indicate this, or dwCancelCallWaitingSize set to 1 and dwCancelCallWaitingOffset pointing to an empty string (single NULL byte) may indicate this.

LINEMESSAGE

The LINEMESSAGE structure contains parameter values that specify a change in status of the line that the application currently has open. The lineGetMessage function returns the LINEMESSAGE structure.

Structure Details

```
typedef struct linemessage_tag {
  DWORD  hDevice;
  DWORD  dwMessageID;
  DWORD_PTR  dwCallbackInstance;
  DWORD_PTR  dwParam1;
  DWORD_PTR  dwParam2;
  DWORD_PTR  dwParam3;
} LINEMESSAGE, FAR *LPLINEMESSAGE;
```

Members

Members	Values
hDevice	A handle to either a line device or a call. The context that dwMessageID provides can determine the nature of this handle (line handle or call handle).
dwMessageID	A line or call device message.
dwCallbackInstance	Instance data passed back to the application, which the application in the dwCallBackInstance parameter of lineInitializeEx specified. TAPI does not interpret this DWORD.
dwParam1	A parameter for the message.
dwParam2	A parameter for the message.
dwParam3	A parameter for the message.

For details about the parameter values that are passed in this structure, see TAPI Line Messages, on page 195.

LINEMONITORTONE

The LINEMONITORTONE structure defines a tone for the purpose of detection. Use this as an entry in an array. An array of tones gets passed to the lineMonitorTones function that monitors these tones and sends a LINE_MONITORTONE message to the application when a detection is made.

A tone with all frequencies set to zero corresponds to silence. An application can thus monitor the call information stream for silence.

Note You must not extend this structure.

Structure Details

```
typedef struct linemonitortone_tag {  DWORD  dwAppSpecific;
  DWORD  dwDuration;
  DWORD  dwFrequency1;
  DWORD  dwFrequency2;
  DWORD  dwFrequency3;
} LINEMONITORTONE, FAR *LPLINEMONITORTONE;
```

Members

Members	Values
dwAppSpecific	Used by the application for tagging the tone. When this tone is detected, the value of the dwAppSpecific member gets passed back to the application.

Members	Values
dwDuration	The duration, in milliseconds, during which the tone should be present before a detection is made.
dwFrequency1	dwFrequency2
dwFrequency3	The frequency, in hertz, of a component of the tone. If fewer than three frequencies are needed in the tone, a value of 0 should be used for the unused frequencies. A tone with all three frequencies set to zero gets interpreted as silence and can be used for silence detection.

LINEPROVIDERENTRY

The LINEPROVIDERENTRY structure provides the information for a single service provider entry. An array of these structures gets returned as part of the LINEPROVIDERLIST structure that the function lineGetProviderList returns.

Note You cannot extend this structure.

Structure Details

```
typedef struct lineproviderentry_tag {
  DWORD  dwPermanentProviderID;
  DWORD  dwProviderFilenameSize;
  DWORD  dwProviderFilenameOffset;
} LINEPROVIDERENTRY, FAR *LPLINEPROVIDERENTRY;
```

Members

Members	Values
dwPermanentProviderID	The permanent provider identifier of the entry.
dwProviderFilenameSizedwProviderFilenameOffset	The size, in bytes, and the offset, in bytes, from the beginning of the LINEPROVIDERLIST structure of a null-terminated string that contains the filename (path) of the service provider DLL (.TSP) file.

LINEPROVIDERLIST

The LINEPROVIDERLIST structure describes a list of service providers. The lineGetProviderList function returns a structure of this type. The LINEPROVIDERLIST structure can contain an array of LINEPROVIDERENTRY structures.

Note You must not extend this structure.

Structure Details

```
typedef struct lineproviderlist_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
DWORD  dwNumProviders;
  DWORD  dwProviderListSize;
  DWORD  dwProviderListOffset;
} LINEPROVIDERLIST, FAR *LPLINEPROVIDERLIST;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that are allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.
dwNumProviders	The number of LINEPROVIDERENTRY structures that are present in the array that is denominated by dwProviderListSize and dwProviderListOffset.
dwProviderListSize dwProviderListOffset	The size, in bytes, and the offset, in bytes, from the beginning of this data structure of an array of LINEPROVIDERENTRY elements, which provide the information on each service provider.

LINEREQMAKECALL

The LINEREQMAKECALL structure describes a request that a call initiated to the lineGetRequest function.

Note You cannot extend this structure.

Structure Details

```
typedef struct linereqmakecall_tag {
  char  szDestAddress[TAPIMAXDESTADDRESSSIZE];
  char  szAppName[TAPIMAXAPPNAMESIZE];
  char  szCalledParty[TAPIMAXCALLEDPARTYSIZE];
```

```
  char  szComment[TAPIMAXCOMMENTSIZE];
} LINEREQMAKECALL, FAR *LPLINEREQMAKECALL;
```

Members

Members	Values
szDestAddress [TAPIMAXADDRESSSIZE]	The null-terminated destination address of the make-call request. The address uses the canonical address format or the dialable address format. The maximum length of the address specifies TAPIMAXDESTADDRESSSIZE characters, which include the NULL terminator. Longer strings get truncated.
szAppName [TAPIMAXAPPNAMESIZE]	The null-terminated, user-friendly application name or filename of the application that originated the request. The maximum length of the address specifies TAPIMAXAPPNAMESIZE characters, which include the NULL terminator.
szCalledParty [TAPIMAXCALLEDPARTYSIZE]	The null-terminated, user-friendly called-party name. The maximum length of the called-party information specifies TAPIMAXCALLEDPARTYSIZE characters, which include the NULL terminator.
szComment [TAPIMAXCOMMENTSIZE]	The null-terminated comment about the call request. The maximum length of the comment string specifies TAPIMAXCOMMENTSIZE characters, which include the NULL terminator.

LINETRANSLATECAPS

The LINETRANSLATECAPS structure describes the address translation capabilities. This structure can contain an array of LINELOCATIONENTRY structures and an array of LINECARDENTRY structures. the lineGetTranslateCaps function returns the LINETRANSLATECAPS structure.

Note You must not extend this structure.

Structure Details

```
typedef struct linetranslatecaps_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwNumLocations;
  DWORD  dwLocationListSize;
  DWORD  dwLocationListOffset;
  DWORD  dwCurrentLocationID;
  DWORD  dwNumCards;
  DWORD  dwCardListSize;
  DWORD  dwCardListOffset;
```

```
  DWORD  dwCurrentPreferredCardID;
} LINETRANSLATECAPS, FAR *LPLINETRANSLATECAPS;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that is allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.
dwNumLocations	The number of entries in the location list. It includes all locations that are defined, including zero (default).
dwLocationListSize dwLocationListOffset	List of locations that are known to the address translation. The list comprises a sequence of LINELOCATIONENTRY structures. The dwLocationListOffset member points to the first byte of the first LINELOCATIONENTRY structure, and the dwLocationListSize member indicates the total number of bytes in the entire list.
dwCurrentLocationID	The dwPermanentLocationID member from the LINELOCATIONENTRY structure for the CurrentLocation.
dwNumCards	The number of entries in the CardList.
dwCardListSize dwCardListOffset	List of calling cards that are known to the address translation. It includes only non-hidden card entries and always includes card 0 (direct dial). The list comprises a sequence of LINECARDENTRY structures. The dwCardListOffset member points to the first byte of the first LINECARDENTRY structure, and the dwCardListSize member indicates the total number of bytes in the entire list.
dwCurrentPreferredCardID	The dwPreferredCardID member from the LINELOCATIONENTRY structure for the CurrentLocation.

LINETRANSLATEOUTPUT

The LINETRANSLATEOUTPUT structure describes the result of an address translation. The lineTranslateAddress function uses this structure.

Note You must not extend this structure.

Structure Details

```
typedef struct linetranslateoutput_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwDialableStringSize;
  DWORD  dwDialableStringOffset;
  DWORD  dwDisplayableStringSize;
  DWORD  dwDisplayableStringOffset;
  DWORD  dwCurrentCountry;
  DWORD  dwDestCountry;
  DWORD  dwTranslateResults;
} LINETRANSLATEOUTPUT, FAR *LPLINETRANSLATEOUTPUT;
```

Members

Members	Values
dwTotalSize	The total size, in bytes, that is allocated to this data structure.
dwNeededSize	The size, in bytes, for this data structure that is needed to hold all the returned information.
dwUsedSize	The size, in bytes, of the portion of this data structure that contains useful information.
dwDialableStringSize dwDialableStringOffset	Contains the translated output that can be passed to the lineMakeCall, lineDial, or other function that requires a dialable string. The output always comprises a null-terminated string (NULL gets included in the count in dwDialableStringSize). This output string includes ancillary fields such as name and subaddress if they were in the input string. This string may contain private information such as calling card numbers. To prevent inadvertent visibility to unauthorized persons, it should not display to the user.
dwDisplayableStringSize dwDisplayableStringOffset	Contains the translated output that can display to the user for confirmation. Identical to DialableString, except the “friendly name” of the card enclosed within bracket characters (for example, “[AT&T Card]”) replaces calling card digits. The ancillary fields, such as name and subaddress, get removed. You can display this string in call-status dialog boxes without exposing private information to unauthorized persons. You can also include this information in call logs.
dwCurrentCountry	Contains the country code that is configured in CurrentLocation. Use this value to control the display by the application of certain user interface elements for local call progress tone detection and for other purposes.

Members	Values
dwDestCountry	Contains the destination country code of the translated address. This value may pass to the dwCountryCode parameter of lineMakeCall and other dialing functions (so the call progress tones of the destination country or region such as a busy signal are properly detected). This field gets set to zero if the destination address that is passed to lineTranslateAddress is not in canonical format.
dwTranslateResults	Indicates the information that is derived from the translation process, which may assist the application in presenting user-interface elements. This field uses one LINETRANSLATERESULT_.

TAPI Phone Functions

TAPI phone functions enable an application to control physical aspects of a phone

Table 15: TAPI Phone Functions

TAPI phone functions
phoneCallbackFunc, on page 271
phoneClose, on page 272
phoneDevSpecific, on page 272
phoneGetDevCaps, on page 272
phoneGetDisplay, on page 273
phoneGetLamp, on page 274
phoneGetMessage, on page 274
phoneGetRing, on page 275
phoneGetStatus, on page 276
phoneGetStatusMessages, on page 277
phoneInitialize, on page 278
phoneInitializeEx, on page 279
phoneNegotiateAPIVersion, on page 281
phoneOpen, on page 282
phoneSetDisplay, on page 283

TAPI phone functions
phoneSetStatusMessages, on page 284
phoneShutdown, on page 286

phoneCallbackFunc

The phoneCallbackFunc function provides a placeholder for the application-supplied function name.

All callbacks occur in the application context. The callback function must reside in a dynamic-link library (DLL) or application module and be exported in the module-definition file.

Function Details

```
VOID FAR PASCAL phoneCallbackFunc(
  HANDLE hDevice,
  DWORD dwMsg,
  DWORD dwCallbackInstance,
  DWORD dwParam1,
  DWORD dwParam2,
  DWORD dwParam3
);
```

Parameters

hDevice

A handle to a phone device that is associated with the callback.

dwMsg

A line or call device message.

dwCallbackInstance

Callback instance data that is passed to the application in the callback. TAPI does not interpret this DWORD.

dwParam1

A parameter for the message.

dwParam2

A parameter for the message.

dwParam3

A parameter for the message.

Further Details

For more information about the parameters that are passed to this callback function, see TAPI Line Messages, on page 195 and TAPI Phone Messages, on page 286.

phoneClose

The phoneClose function closes the specified open phone device.

Function Details

```
LONG phoneClose(
  HPHONE hPhone
);
```

Parameter

hPhone

A handle to the open phone device that is to be closed. If the function succeeds, this means that the handle is no longer valid.

phoneDevSpecific

The phoneDevSpecific function gets used as a general extension mechanism to enable a telephony API implementation to provide features that are not described in the other TAPI functions. The meanings of these extensions are device specific.

When used with the Cisco Unified TSP, you can use phoneDevSpecific to send device-specific data to a phone device.

Function Details

```
LONG WINAPI phoneDevSpecific (
HPHONE hPhone,
LPVOID lpParams,
DWORD dwSize
);
```

Parameters

hPhone

A handle to a phone device.

lpParams

A pointer to a memory area used to hold a parameter block. Its interpretation is device specific. TAPI passes the contents of the parameter block unchanged to or from the service provider.

dwSize

The size in bytes of the parameter block area.

phoneGetDevCaps

The phoneGetDevCaps function queries a specified phone device to determine its telephony capabilities.

Function Details

```
LONG phoneGetDevCaps(
  HPHONEAPP hPhoneApp,
  DWORD dwDeviceID,
  DWORD dwAPIVersion,
  DWORD dwExtVersion,
  LPPHONECAPS lpPhoneCaps
);
```

Parameters

hPhoneApp

The handle to the registration with TAPI for this application.

dwDeviceID

The phone device that is to be queried.

dwAPIVersion

The version number of the telephony API that is to be used. The high-order word contains the major version number; the low-order word contains the minor version number. You can obtain this number with the function phoneNegotiateAPIVersion.

dwExtVersion

The version number of the service provider-specific extensions to be used. This number is obtained with the function phoneNegotiateExtVersion. It can be left as zero if no device-specific extensions are to be used. Otherwise, the high-order word contains the major version number, the low-order word contains the minor version number.

lpPhoneCaps

A pointer to a variably sized structure of type PHONECAPS. Upon successful completion of the request, this structure is filled with phone device capabilities information.

phoneGetDisplay

The phoneGetDisplay function returns the current contents of the specified phone display.

Function Details

```
LONG phoneGetDisplay(
  HPHONE hPhone,
  LPVARSTRING lpDisplay
);
```

Parameters

hPhone

A handle to the open phone device.

lpDisplay

A pointer to the memory location where the display content is to be stored, of type VARSTRING.

phoneGetLamp

The phoneGetLamp function returns the current lamp mode of the specified lamp.

Note Cisco Unified IP Phones 79xx series do not support this function.

Function Details

```
LONG phoneGetLamp(
  HPHONE hPhone,
  DWORD dwButtonLampID,
  LPDWORD lpdwLampMode
);
```

Parameters

hPhone

A handle to the open phone device.

dwButtonLampID

The identifier of the lamp that is to be queried. See PHONE_BUTTON, on page 287 for lamp IDs.

lpdwLampMode

Note Cisco Unified IP Phones 79xx series do not support this function.

A pointer to a memory location that holds the lamp mode status of the given lamp. The lpdwLampMode parameter can have at most one bit set. This parameter uses the following PHONELAMPMODE_constants:

- PHONELAMPMODE_FLASH -Flash means slow on and off.
- PHONELAMPMODE_FLUTTER -Flutter means fast on and off.
- PHONELAMPMODE_OFF -The lamp is off.
- PHONELAMPMODE_STEADY -The lamp is continuously lit.
- PHONELAMPMODE_WINK -The lamp winks.
- PHONELAMPMODE_UNKNOWN -The lamp mode is currently unknown.
- PHONELAMPMODE_DUMMY -Use this value to describe a button/lamp position that has no corresponding lamp.

phoneGetMessage

The phoneGetMessage function returns the next TAPI message that is queued for delivery to an application that is using the Event Handle notification mechanism (see phoneInitializeEx for further details).

Function Details

```
LONG WINAPI phoneGetMessage(
  HPHONEAPP hPhoneApp,
  LPPHONEMESSAGE lpMessage,
  DWORD dwTimeout
);
```

Parameters

hPhoneApp

The handle that phoneInitializeEx returns. The application must have set the PHONEINITIALIZEEXOPTION_USEEVENT option in the dwOptions member of the PHONEINITIALIZEEXPARAMS structure.

lpMessage

A pointer to a PHONEMESSAGE structure. Upon successful return from this function, the structure contains the next message that had been queued for delivery to the application.

dwTimeout

The time-out interval, in milliseconds. The function returns if the interval elapses, even if no message can be returned. If dwTimeout is zero, the function checks for a queued message and returns immediately. If dwTimeout is INFINITE, the time-out interval never elapses.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALAPPHANDLE, PHONEERR_OPERATIONFAILED, PHONEERR_INVALPOINTER, PHONEERR_NOMEM.

phoneGetRing

The phoneGetRing function enables an application to query the specified open phone device as to its current ring mode.

Function Details

```
LONG phoneGetRing(
  HPHONE hPhone,
  LPDWORD lpdwRingMode,
  LPDWORD lpdwVolume
);
```

Parameters

hPhone

A handle to the open phone device.

lpdwRingMode

The ringing pattern with which the phone is ringing. Zero indicates that the phone is not ringing.

The system supports four ring modes.

The following table lists the valid ring modes.

Table 16: Ring Modes

Ring Modes	Definition
0	Off
1	Inside Ring
2	Outside Ring
3	Feature Ring

lpdwVolume

The volume level with which the phone is ringing. This parameter has no meaning; the value 0x8000 always gets returned.

phoneGetStatus

The phoneGetStatus function enables an application to query the specified open phone device for its overall status.

Function Details

```
LONG WINAPI phoneGetStatusMessages(
HPHONE hPhone,
LPPHONESTATUS lpPhoneStatus
) ;
```

Parameters

hPhone

A handle to the open phone device to be queried.

lpPhoneStatus

A pointer to a variably sized data structure of type PHONESTATUS, which is loaded with the returned information about the phone status.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Return values include the following:

PHONEERR_INVALPHONEHANDLE, PHONEERR_NOMEM PHONEERR_INVALPOINTER, PHONEERR_RESOURCEUNAVAIL PHONEERR_OPERATIONFAILED, PHONEERR_STRUCTURETOOSMALL PHONEERR_OPERATIONUNAVAIL, PHONEERR_UNINITIALIZED

phoneGetStatusMessages

The phoneGetStatusMessages function returns information about which phone-state changes on the specified phone device generate a callback to the application.

An application can use phoneGetStatusMessages to query the generation of the corresponding messages. The phoneSetStatusMessages can control Message generation. All phone status messages remain disabled by default.

Function Details

```
LONG WINAPI phoneGetStatusMessages(
  HPHONE hPhone,
  LPDWORD lpdwPhoneStates,
  LPDWORD lpdwButtonModes,
  LPDWORD lpdwButtonStates
);
```

Parameters

hPhone

A handle to the open phone device that is to be monitored.

lpdwPhoneStates

A pointer to a DWORD holding zero, one or more of the PHONESTATE_Constants. These flags specify the set of phone status changes and events for which the application can receive notification messages. You can enable or disable monitoring individually for the following states:

- PHONESTATE_OTHER
- PHONESTATE_CONNECTED
- PHONESTATE_DISCONNECTED
- PHONESTATE_OWNER
- PHONESTATE_MONITORS
- PHONESTATE_DISPLAY
- PHONESTATE_LAMP
- PHONESTATE_RINGMODE
- PHONESTATE_RINGVOLUME
- PHONESTATE_HANDSETHOOKSWITCH
- PHONESTATE_HANDSETVOLUME
- PHONESTATE_HANDSETGAIN
- PHONESTATE_SPEAKERHOOKSWITCH
- PHONESTATE_SPEAKERVOLUME
- PHONESTATE_SPEAKERGAIN

- PHONESTATE_HEADSETHOOKSWITCH
- PHONESTATE_HEADSETVOLUME
- PHONESTATE_HEADSETGAIN
- PHONESTATE_SUSPEND
- PHONESTATE_RESUMEF
- PHONESTATE_DEVSPECIFIC
- PHONESTATE_REINIT
- PHONESTATE_CAPSCHANGE
- PHONESTATE_REMOVED

lpdwButtonModes

A pointer to a DWORD that contains flags that specify the set of phone-button modes for which the application can receive notification messages. This parameter uses zero, one, or more of the PHONEBUTTONMODE_Constants.

lpdwButtonStates

A pointer to a DWORD that contains flags that specify the set of phone button state changes for which the application can receive notification messages. This parameter uses zero, one, or more of the PHONEBUTTONSTATE_ Constants.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALPHONEHANDLE

PHONEERR_NOMEM

PHONEERR_INVALPOINTER

PHONEERR_RESOURCEUNAVAIL

PHONEERR_OPERATIONFAILED

PHONEERR_UNINITIALIZED.

phoneInitialize

Although the phoneInitialize function is obsolete, tapi.dll and tapi32.dll continue to export it for backward compatibility with applications that are using TAPI versions 1.3 and 1.4.

Function Details

```
LONG WINAPI phoneInitialize(
  LPHPHONEAPP lphPhoneApp,
  HINSTANCE hInstance,
  PHONECALLBACK lpfnCallback,
  LPCSTR lpszAppName,
```

```
  LPDWORD lpdwNumDevs
);
```

Parameters

lphPhoneApp

A pointer to a location that is filled with the application usage handle for TAPI.

hInstance

The instance handle of the client application or DLL.

lpfnCallback

The address of a callback function that is invoked to determine status and events on the phone device.

lpszAppName

A pointer to a null-terminated string that contains displayable characters. If this parameter is non-NULL, it contains an application-supplied name of the application. This name, which is provided in the PHONESTATUS structure, indicates, in a user-friendly way, which application is the current owner of the phone device. You can use this information for logging and status reporting purposes. If lpszAppName is NULL, the application filename gets used instead.

lpdwNumDevs

A pointer to DWORD. This location gets loaded with the number of phone devices that are available to the application.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALAPPNAME

PHONEERR_INIFILECORRUPT

PHONEERR_INVALPOINTER

PHONEERR_NOMEM

PHONEERR_OPERATIONFAILED

PHONEERR_REINIT

PHONEERR_RESOURCEUNAVAIL

PHONEERR_NODEVICE

PHONEERR_NODRIVER

PHONEERR_INVALPARAM

phoneInitializeEx

The phoneInitializeEx function initializes the application use of TAPI for subsequent use of the phone abstraction. It registers the application specified notification mechanism and returns the number of phone devices that are available to the application. A phone device represents any device that provides an implementation for the phone-prefixed functions in the telephony API.

Function Details

```
LONG WINAPI phoneInitializeEx(
  LPHPHONEAPP lphPhoneApp,
  HINSTANCE hInstance,
  PHONECALLBACK lpfnCallback,
  LPCSTR lpszFriendlyAppName,
  LPDWORD lpdwNumDevs,
  LPDWORD lpdwAPIVersion,
  LPPHONEINITIALIZEEXPARAMS lpPhoneInitializeExParams
);
```

Parameters

lphPhoneApp

A pointer to a location that is filled with the application usage handle for TAPI.

hInstance

The instance handle of the client application or DLL. The application or DLL can pass NULL for this parameter, in which case TAPI uses the module handle of the root executable of the process.

lpfnCallback

The address of a callback function that is invoked to determine status and events on the line device, addresses, or calls, when the application is using the "hidden window" method of event notification (for more information, see phoneCallbackFunc). When the application chooses to use the event handle or completion port event notification mechanisms, this parameter gets ignored and should be set to NULL.

lpszFriendlyAppName

A pointer to a null-terminated string that contains only displayable characters. If this parameter is not NULL, it contains an application-supplied name for the application. This name, which is provided in the PHONESTATUS structure, indicates, in a user-friendly way, which application has ownership of the phone device. If lpszFriendlyAppName is NULL, the application module filename gets used instead (as returned by the Windows function GetModuleFileName).

lpdwNumDevs

A pointer to a DWORD. Upon successful completion of this request, the number of phone devices that are available to the application fills this location.

lpdwAPIVersion

A pointer to a DWORD. The application must initialize this DWORD, before calling this function, to the highest API version that it is designed to support (for example, the same value that it would pass into dwAPIHighVersion parameter of phoneNegotiateAPIVersion). Do no use artificially high values; ensure the values are accurately set. TAPI translates any newer messages or structures into values or formats that the application version supports. Upon successful completion of this request, the highest API version that TAPI supports fills this location, which allows the application to detect and adapt to being installed on a system with an older version of TAPI.

lpPhoneInitializeExParams

A pointer to a structure of type PHONEINITIALIZEEXPARAMS that contains additional parameters that are used to establish the association between the application and TAPI (specifically, the application-selected event notification mechanism and associated parameters).

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALAPPNAME

PHONEERR_OPERATIONFAILED

PHONEERR_INIFILECORRUPT

PHONEERR_INVALPOINTER

PHONEERR_REINIT

PHONEERR_NOMEM

PHONEERR_INVALPARAM

phoneNegotiateAPIVersion

Use the phoneNegotiateAPIVersion function to negotiate the API version number to be used with the specified phone device. It returns the extension identifier that the phone device supports, or zeros if no extensions are provided.

Function Details

```
LONG WINAPI phoneNegotiateAPIVersion(
  HPHONEAPP hPhoneApp,
  DWORD dwDeviceID,
  DWORD dwAPILowVersion,
  DWORD dwAPIHighVersion,
  LPDWORD lpdwAPIVersion,
  LPPHONEEXTENSIONID lpExtensionID
);
```

Parameters

hPhoneApp

The handle to the application registration with TAPI.

dwDeviceID

The phone device to be queried.

dwAPILowVersion

The least recent API version with which the application is compliant. The high-order word represents the major version number, and the low-order word represents the minor version number.

dwAPIHighVersion

The most recent API version with which the application is compliant. The high-order word represents the major version number, and the low-order word represents the minor version number.

lpdwAPIVersion

A pointer to a DWORD in which the API version number that was negotiated will be returned. If negotiation succeeds, this number ranges from dwAPILowVersion to dwAPIHighVersion.

lpExtensionID

A pointer to a structure of type PHONEEXTENSIONID. If the service provider for the specified dwDeviceID parameter supports provider-specific extensions, this structure gets filled with the extension identifier of these extensions when negotiation succeeds. This structure contains all zeros if the line provides no extensions. An application can ignore the returned parameter if it does not use extensions.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALAPPHANDLE

PHONEERR_OPERATIONFAILED

PHONEERR_BADDEVICEID

PHONEERR_OPERATIONUNAVAIL

PHONEERR_NODRIVER

PHONEERR_NOMEM

PHONEERR_INVALPOINTER

PHONEERR_RESOURCEUNAVAIL,PHONEERR_INCOMPATIBLEAPIVERSION

PHONEERR_UNINITIALIZED

PHONEERR_NODEVICE

phoneOpen

The phoneOpen function opens the specified phone device. Open the device by using either owner privilege or monitor privilege. An application that opens the phone with owner privilege can control the lamps, display, ringer, and hookswitch or hookswitches that belong to the phone. An application that opens the phone device with monitor privilege receives notification only about events that occur at the phone, such as hookswitch changes or button presses. Because ownership of a phone device is exclusive, only one application at a time can have a phone device opened with owner privilege. The phone device can, however, be opened multiple times with monitor privilege.

Note To open a phone device on a CTI port, first ensure a corresponding line device is open.

Function Details

```
LONG phoneOpen(
  HPHONEAPP hPhoneApp,
  DWORD dwDeviceID,
  LPHPHONE lphPhone,
  DWORD dwAPIVersion,
  DWORD dwExtVersion,
  DWORD dwCallbackInstance,
  DWORD dwPrivilege
);
```

Parameters

hPhoneApp

A handle by which the application is registered with TAPI.

dwDeviceID

The phone device to be opened.

lphPhone

A pointer to an HPHONE handle that identifies the open phone device. Use this handle to identify the device when invoking other phone control functions.

dwAPIVersion

The API version number under which the application and telephony API agreed to operate. Obtain this number from phoneNegotiateAPIVersion.

dwExtVersion

The extension version number under which the application and the service provider agree to operate. This number is zero if the application does not use any extensions. Obtain this number from phoneNegotiateExtVersion.

dwCallbackInstance

User instance data that is passed back to the application with each message. The telephony API does not interpret this parameter.

dwPrivilege

The privilege requested. The dwPrivilege parameter can have only one bit set. This parameter uses the following PHONEPRIVILEGE_constants:

- PHONEPRIVILEGE_MONITOR -An application that opens a phone device with this privilege gets informed about events and state changes that occur on the phone. The application cannot invoke any operations on the phone device that would change its state.
- PHONEPRIVILEGE_OWNER -An application that opens a phone device in this mode can change the state of the lamps, ringer, display, and hookswitch devices of the phone. Having owner privilege to a phone device automatically includes monitor privilege as well.

phoneSetDisplay

The phoneSetDisplay function causes the specified string to display on the specified open phone device.

Note Prior to Release 4.0, Cisco Unified Communications Manager messages that were passed to the phone would automatically overwrite any messages sent to the phone by using phoneSetDisplay(). In Cisco Unified Communications Manager 4.0, the message sent to the phone in the phoneSetDisplay() API remains on the phone until the phone is rebooted. If the application wants to clear the text from the display and see the Cisco Unified Communications Manager messages again, a NULL string, not spaces, should be passed in the phoneSetDisplay() API. In other words, the lpsDisplay parameter should be NULL and the dwSize should be set to 0.

Function Details

```
LONG phoneSetDisplay(
  HPHONE hPhone,
  DWORD dwRow,
  DWORD dwColumn,
  LPCSTR lpsDisplay,
  DWORD dwSize
);
```

Parameters

hPhone

A handle to the open phone device. The application must be the owner of the phone.

dwRow

The row position on the display where the new text displays.

dwColumn

The column position on the display where the new text displays.

lpsDisplay

A pointer to the memory location where the display content is stored. The display information must follow the format that is specified in the dwStringFormat member of the device capabilities for this phone.

dwSize

The size in bytes of the information to which lpsDisplay points.

phoneSetStatusMessages

The phoneSetStatusMessages function enables an application to monitor the specified phone device for selected status events.

See TAPI Phone Messages, on page 286 for supported messages.

Function Details

```
LONG phoneSetStatusMessages(
  HPHONE hPhone,
  DWORD dwPhoneStates,
  DWORD dwButtonModes,
  DWORD dwButtonStates
);
```

Parameters

hPhone

A handle to the open phone device to be monitored.

dwPhoneStates

These flags specify the set of phone status changes and events for which the application can receive notification messages. This parameter can have zero, one, or more bits set. This parameter uses the following PHONESTATE_ constants:

- PHONESTATE_OTHER -Phone status items other than those in the following list changed. The application should check the current phone status to determine which items changed.
- PHONESTATE_OWNER -The number of owners for the phone device changed.
- PHONESTATE_MONITORS -The number of monitors for the phone device changed.
- PHONESTATE_DISPLAY -The display of the phone changed.
- PHONESTATE_LAMP -A lamp of the phone changed.
- PHONESTATE_RINGMODE -The ring mode of the phone changed.
- PHONESTATE_SPEAKERHOOKSWITCH -The hookswitch state changed for this speakerphone.
- PHONESTATE_REINIT -Items changed in the configuration of phone devices. To become aware of these changes (as with the appearance of new phone devices), the application should reinitialize its use of TAPI. New phoneInitialize, phoneInitializeEx, and phoneOpen requests get denied until applications have shut down their usage of TAPI. The hDevice parameter of the PHONE_STATE message stays NULL for this state change because it applies to any line in the system. Because of the critical nature of PHONESTATE_REINIT, you cannot mask such messages, so the setting of this bit gets ignored, and the messages always get delivered to the application.
- PHONESTATE_REMOVED -Indicates that the service provider is removing the device from the system (most likely through user action, through a control panel or similar utility). A PHONE_CLOSE message on the device immediately follows a PHONE_STATE message with this value. Subsequent attempts to access the device prior to TAPI being reinitialized result in PHONEERR_NODEVICE being returned to the application. If a service provider sends a PHONE_STATE message that contains this value to TAPI, TAPI passes it along to applications that negotiated TAPI version 1.4 or later; applications that negotiated a previous TAPI version do not receive any notification.

dwButtonModes

The set of phone-button modes for which the application can receive notification messages. This parameter can have zero, one, or more bits set. This parameter uses the following PHONEBUTTONMODE_constants:

- PHONEBUTTONMODE_CALL -The button is assigned to a call appearance.
- PHONEBUTTONMODE_FEATURE -The button is assigned to requesting features from the switch, such as hold, conference, and transfer.
- PHONEBUTTONMODE_KEYPAD -The button is one of the 12 keypad buttons, ‘0’ through ‘9’, ‘*’, and ‘#’.
- PHONEBUTTONMODE_DISPLAY -The button is a “soft” button that is associated with the phone display. A phone set can have zero or more display buttons.

dwButtonStates

The set of phone-button state changes for which the application can receive notification messages. If the dwButtonModes parameter is zero, the system ignores dwButtonStates. If dwButtonModes has one or

more bits set, this parameter also must have at least one bit set. This parameter uses the following PHONEBUTTONSTATE_constants:

- PHONEBUTTONSTATE_UP -The button is in the “up” state.
- PHONEBUTTONSTATE_DOWN -The button is in the “down” state (pressed down).
- PHONEBUTTONSTATE_UNKNOWN -The up or down state of the button is unknown at this time but may become known later.
- PHONEBUTTONSTATE_UNAVAIL -The service provider does not know the up or down state of the button, and the state will not become known.

phoneShutdown

The phoneShutdown function shuts down the application usage of the TAPI phone abstraction.

Note If this function is called when the application has open phone devices, these devices are closed.

Function Details

```
LONG WINAPI phoneShutdown(
  HPHONEAPP hPhoneApp
);
```

Parameter

hPhoneApp

The application usage handle for TAPI.

Return Values

Returns zero if the request succeeds or a negative number if an error occurs. Possible return values follow:

PHONEERR_INVALAPPHANDLE, PHONEERR_NOMEM, PHONEERR_UNINITIALIZED, PHONEERR_RESOURCEUNAVAIL.

TAPI Phone Messages

Messages notify the application of asynchronous events. All messages get sent to the application through the message notification mechanism that the application specified in lineInitializeEx. The message always contains a handle to the relevant object (phone, line, or call), of which the application can determine the type from the message type. The following table describes TAPI Phone messages.

Table 17: TAPI Phone Messages

TAPI Phone Messages
PHONE_BUTTON, on page 287

TAPI Phone Messages
PHONE_CLOSE, on page 290
PHONE_CREATE, on page 290
PHONE_REMOVE, on page 291
PHONE_REPLY, on page 292
PHONE_STATE, on page 292

PHONE_BUTTON

The PHONE_BUTTON message notifies the application that button press monitoring is enabled if it has detected a button press on the local phone.

Function Details

```
PHONE_BUTTON
hPhone = (HPHONE) hPhoneDevice;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) idButtonOrLamp;
dwParam2 = (DWORD) ButtonMode;
dwParam3 = (DWORD) ButtonState;
```

Parameters

hPhone

A handle to the phone device.

dwCallbackInstance

The callback instance that is provided when the phone device for this application is opened.

dwParam1

The button/lamp identifier of the button that was pressed. Button identifiers zero through 11 always represent the KEYPAD buttons, with ‘0’ being button identifier zero, ‘1’ being button identifier 1 (and so on through button identifier 9), and with ‘*’ being button identifier 10, and ‘#’ being button identifier 11. Find additional information about a button identifier with phoneGetDevCaps, on page 272.

dwParam2

The button mode of the button. The button mode for each button ID gets listed as shown in the table below.

The TAPI service provider cannot detect button down or button up state changes. To conform to the TAPI specification, two messages are sent to simulate a down state followed by an up state in dwparam3.

This parameter uses the following PHONEBUTTONMODE_constants:

- PHONEBUTTONMODE_CALL -The button is assigned to a call appearance.

- PHONEBUTTONMODE_FEATURE -The button is assigned to requesting features from the switch, such as hold, conference, and transfer.
- PHONEBUTTONMODE_KEYPAD -The button is one of the 12 keypad buttons, ‘0’ through ‘9’, ‘*’, and ‘#’.
- PHONEBUTTONMODE_DISPLAY -The button is a soft button that is associated with the phone display. A phone set can have zero or more display buttons.

dwParam3

Specifies whether this is a button-down event or a button-up event. This parameter uses the following PHONEBUTTONSTATE_constants:

- PHONEBUTTONSTATE_UP -The button is in the up state.
- PHONEBUTTONSTATE_DOWN -The button is in the down state (pressed down).
- PHONEBUTTONSTATE_UNKNOWN -The up or down state of the button is not known at this time and may be known later.
- PHONEBUTTONSTATE_UNAVAIL -The service provider does not know the up or down state of the button, and the state cannot become known at a future time.

Button ID values of zero through 11 map to the keypad buttons as defined by TAPI. Values above 11 map to line and feature buttons. The low-order part of the DWORD specifies the feature. The high-order part of the DWORD specifies the instance number of that feature. The following table lists all possible values for the low-order part of the DWORD that corresponds to the feature.

Use the following expression to make the button ID:

ButtonID = (instance << 16) | featureID

The following table lists the valid phone button values.

Table 18: Phone Button Values

Value	Feature	Has Instance	Button Mode
0	Keypad button 0	No	Keypad
1	Keypad button 1	No	Keypad
2	Keypad button 2	No	Keypad
3	Keypad button 3	No	Keypad
4	Keypad button 4	No	Keypad
5	Keypad button 5	No	Keypad
6	Keypad button 6	No	Keypad
7	Keypad button 7	No	Keypad
8	Keypad button 8	No	Keypad
9	Keypad button 9	No	Keypad

Value	Feature	Has Instance	Button Mode
10	Keypad button '*'	No	Keypad
11	Keypad button '#'	No	Keypad
12	Last Number Redial	No	Feature
13	Speed Dial	Yes	Feature
14	Hold	No	Feature
15	Transfer	No	Feature
16	Forward All (for line one)	No	Feature
17	Forward Busy (for line one)	No	Feature
18	Forward No Answer (for line one)	No	Feature
19	Display	No	Feature
20	Line	Yes	Call
21	Chat (for line one)	No	Feature
22	Whiteboard (for line one)	No	Feature
23	Application Sharing (for line one)	No	Feature
24	T120 File Transfer (for line one)	No	Feature
25	Video (for line one)	No	Feature
26	Voice Mail (for line one)	No	Feature
27	Answer Release	No	Feature
28	Auto-answer	No	Feature
44	Generic Custom Button 1	Yes	Feature
45	Generic Custom Button 2	Yes	Feature
46	Generic Custom Button 3	Yes	Feature
47	Generic Custom Button 4	Yes	Feature
48	Generic Custom Button 5	Yes	Feature

PHONE_CLOSE

The PHONE_CLOSE message gets sent when an open phone device is forcibly closed as part of resource reclamation. The device handle is no longer valid after this message is sent.

Function Details

```
PHONE_CLOSE
hPhone = (HPHONE) hPhoneDevice;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) 0;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hPhone

A handle to the open phone device that was closed. The handle is no longer valid after this message is sent.

dwCallbackInstance

The callback instance of the application that is provided on an open phone device.

dwParam1
Not used.
dwParam2
Not used.
dwParam3
Not used.

PHONE_CREATE

The PHONE_CREATE message gets sent to inform applications of the creation of a new phone device.

Note CTI Manager cluster support, extension mobility, change notification, and user addition to the directory can generate PHONE_CREATE events.

Function Details

```
PHONE_CREATE
hPhone = (HPHONE) hPhoneDevice;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) idDevice;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hPhone

Not used.

dwCallbackInstance
Not used.
dwParam1

The dwDeviceID of the newly created device.

dwParam2
Not used.
dwParam3
Not used.

PHONE_REMOVE

The PHONE_REMOVE message gets sent to inform an application of the removal (deletion from the system) of a phone device. Generally, this method is not used for temporary removals, such as extraction of PCMCIA devices, but only for permanent removals in which the service provider would no longer report the device, if TAPI were reinitialized.

Note CTI Manager cluster support, extension mobility, change notification, and user deletion from the directory can generate PHONE_REMOVE events.

Function Details

```
PHONE_REMOVE
dwDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) dwDeviceID;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

dwDevice
Reserved. Set to zero.
dwCallbackInstance
Reserved. Set to zero.
dwParam1

Identifier of the phone device that was removed.

dwParam2
Reserved. Set to zero.
dwParam3
Reserved. Set to zero.

PHONE_REPLY

The TAPI PHONE_REPLY message gets sent to an application to report the results of function call that completed asynchronously.

Function Details

```
PHONE_REPLY
hPhone = (HPHONE) 0;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) idRequest;
dwParam2 = (DWORD) Status;
dwParam3 = (DWORD) 0;
```

Parameters

hPhone
Not used.

dwCallbackInstance

Returns the application callback instance.

dwParam1

The request identifier for which this is the reply.

dwParam2

The success or error indication. The application should cast this parameter into a LONG. Zero indicates success; a negative number indicates an error.

dwParam3
Not used.

PHONE_STATE

TAPI sends the PHONE_STATE message to an application whenever the status of a phone device changes.

Function Details

```
PHONE_STATE
hPhone = (HPHONE) hPhoneDevice;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) PhoneState;
dwParam2 = (DWORD) PhoneStateDetails;
dwParam3 = (DWORD) 0;
```

Parameters

hPhone

A handle to the phone device.

dwCallbackInstance

The callback instance that is provided when the phone device is opened for this application.

dwParam1

The phone state that changed. This parameter uses the following PHONESTATE_constants:

- PHONESTATE_OTHER -Phone-status items other than the following ones changed. The application should check the current phone status to determine which items changed.
- PHONESTATE_CONNECTED -The connection between the phone device and TAPI was just made. This happens when TAPI is first invoked or when the wire that connects the phone to the computer is plugged in while TAPI is active.
- PHONESTATE_DISCONNECTED -The connection between the phone device and TAPI just broke. This happens when the wire that connects the phone set to the computer is unplugged while TAPI is active.
- PHONESTATE_OWNER -The number of owners for the phone device changed.
- PHONESTATE_MONITORS -The number of monitors for the phone device changed.
- PHONESTATE_DISPLAY -The display of the phone changed.
- PHONESTATE_LAMP -A lamp of the phone changed.
- PHONESTATE_RINGMODE -The ring mode of the phone changed.
- PHONESTATE_HANDSETHOOKSWITCH -The hookswitch state changed for this speakerphone.
- PHONESTATE_REINIT -Items changed in the configuration of phone devices. To become aware of these changes (as with the appearance of new phone devices), the application should reinitialize its use of TAPI. The hDevice parameter of the PHONE_STATE message stays NULL for this state change as it applies to any of the phones in the system.
- PHONESTATE_REMOVED -Indicates that the device is being removed from the system by the service provider (most likely through user action, through a control panel or similar utility). Normally, a PHONE_CLOSE message on the device immediately follows a PHONE_STATE message with this value. Subsequent attempts to access the device prior to TAPI being reinitialized result in PHONEERR_NODEVICE being returned to the application. If a service provider sends a PHONE_STATE message that contains this value to TAPI, TAPI passes it along to applications that negotiated TAPI version 1.4 or later; applications that negotiated a previous API version do not receive any notification.
- PHONESTATE_SUSPEND -Indicates the phone unregisters as it enters Energywise DeepSleep/PowersavePlus mode.

dwParam2

Phone state-dependent information that details the status change. This parameter is not used if multiple flags are set in dwParam1 because multiple status items get changed. The application should invoke phoneGetStatus to obtain a complete set of information.

Parameter dwparam2 can comprise one of PHONESTATE_LAMP, PHONESTATE_DISPLAY, PHONESTATE_HANDSETHOOKSWITCH, or PHONESTATE_RINGMODE. Because the Cisco Unified TSP cannot differentiate among hook switches for handsets, headsets, or speaker, the PHONESTATE_HANDSETHOOKSWITCH value always gets used for hook switches.

If dwparam2 is PHONESTATE_LAMP, dwparam2 is the button ID that the PHONE_BUTTON message defines.

If dwParam1 is PHONESTATE_OWNER, dwParam2 contains the new number of owners.

If dwParam1 is PHONESTATE_MONITORS, dwParam2 contains the new number of monitors.

If dwParam1 is PHONESTATE_LAMP, dwParam2 contains the button/lamp identifier of the lamp that changed.

If dwParam1 is PHONESTATE_RINGMODE, dwParam2 contains the new ring mode.

If dwParam1 is PHONESTATE_HANDSET, SPEAKER, or HEADSET, dwParam2 contains the new hookswitch mode of that hookswitch device. This parameter uses the following PHONEHOOKSWITCHMODE_constants:

- PHONEHOOKSWITCHMODE_ONHOOK -The microphone and speaker both remain on hook for this device.
- PHONEHOOKSWITCHMODE_MICSPEAKER -The microphone and speaker both remain active for this device. The Cisco Unified TSP cannot distinguish among handsets, headsets, or speakers, so this value gets sent when the device is off hook.

If dw Param1 is PHONESTATE_SUSPEND, dwParam2 contains the reason EnergyWisePowerSavePlus when the phone unregisters as it enters EnergywiseDeepSleep.

dwParam3

The TAPI specification specifies that dwparam3 is zero; however, the Cisco Unified TSP will send the new lamp state to the application in dwparam3 to avoid the call to phoneGetLamp to obtain the state when dwparam2 is PHONESTATE_LAMP.

TAPI Phone Structures

This section describes the TAPI phone structures that Cisco Unified TSP supports:

.

Table 19: TAPI Phone Structures

TAPI Phone Structure
PHONECAPS Structure, on page 294
PHONEINITIALIZEEXPARAMS, on page 296
PHONEMESSAGE, on page 297
PHONESTATUS, on page 298
VARSTRING, on page 300

PHONECAPS Structure

This section lists the Cisco-set attributes for each member of the PHONECAPS structure. If the value of a structure member is device, line, or call specific, the list gives the value for each condition.

Members

dwProviderInfoSize
dwProviderInfoOffset

"Cisco Unified TSPxxx.TSP: Cisco IP PBX Service Provider Ver. X.X(x.x)" where the text before the colon specifies the file name of the TSP, and the text after "Ver. " specifies the version of the TSP.

dwPhoneInfoSize
dwPhoneInfoOffset

"DeviceType:[type]" where type specifies the device type that is specified in the Cisco Unified Communications Manager database.

dwPermanentPhoneID
dwPhoneNameSize
dwPhoneNameOffset

"Cisco Phone: [deviceName]" where deviceName specifies the name of the device in the Cisco Unified Communications Manager database.

dwStringFormat

STRINGFORMAT_ASCII

dwPhoneStates

PHONESTATE_OWNER |

PHONESTATE_MONITORS |

PHONESTATE_DISPLAY | (Not set for CTI Route Points)

PHONESTATE_LAMP | (Not set for CTI Route Points)

PHONESTATE_RESUME |

PHONESTATE_REINIT |

PHONESTATE_SUSPEND

dwHookSwitchDevs

PHONEHOOKSWITCHDEV_HANDSET (Not set for CTI Route Points)

dwHandsetHookSwitchModes

PHONEHOOKSWITCHMODE_ONHOOK | (Not set for CTI Route Points)

PHONEHOOKSWITCHMODE_MICSPEAKER | (Not set for CTI Route Points)

PHONEHOOKSWITCHMODE_UNKNOWN (Not set for CTI Route Points)

dwDisplayNumRows (Not set for CTI Route Points)
1
dwDisplayNumColumns

20 (Not set for CTI Route Points)

dwNumRingModes

3 (Not set for CTI Route Points)

dwPhoneFeatures (Not set for CTI Route Points)

PHONEFEATURE_GETDISPLAY |

PHONEFEATURE_GETLAMP |

PHONEFEATURE_GETRING |

PHONEFEATURE_SETDISPLAY |

PHONEFEATURE_SETLAMP

dwMonitoredHandsetHookSwitchModes

PHONEHOOKSWITCHMODE_ONHOOK | (Not set for CTI Route Points)

PHONEHOOKSWITCHMODE_MICSPEAKER (Not set for CTI Route Points)

PHONEINITIALIZEEXPARAMS

The PHONEINITIALIZEEXPARAMS structure contains parameters that are used to establish the association between an application and TAPI; for example, the application selected event notification mechanism. The phoneInitializeEx function uses this structure.

Structure Details

```
typedef struct phoneinitializeexparams_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwOptions;
  union
  {
    HANDLE hEvent;
    HANDLE hCompletionPort;
  } Handles;
  DWORD dwCompletionKey;
} PHONEINITIALIZEEXPARAMS, FAR *LPPHONEINITIALIZEEXPARAMS;
```

Members

dwTotalSize

The total size, in bytes, that is allocated to this data structure.

dwNeededSize

The size, in bytes, for this data structure that is needed to hold all the returned information.

dwUsedSize

The size, in bytes, of the portion of this data structure that contains useful information.

dwOptions

One of the PHONEINITIALIZEEXOPTION_Constants. Specifies the event notification mechanism that the application wants to use.

hEvent

If dwOptions specifies PHONEINITIALIZEEXOPTION_USEEVENT, TAPI returns the event handle in this member.

hCompletionPort

If dwOptions specifies PHONEINITIALIZEEXOPTION_USECOMPLETIONPORT, the application must specify, in this member, the handle of an existing completion port that is opened by using CreateIoCompletionPort.

dwCompletionKey

If dwOptions specifies PHONEINITIALIZEEXOPTION_USECOMPLETIONPORT, the application must specify in this field a value that is returned through the lpCompletionKey parameter of GetQueuedCompletionStatus to identify the completion message as a telephony message.

PHONEMESSAGE

The PHONEMESSAGE structure contains the next message that is queued for delivery to the application. The phoneGetMessage function returns the following structure.

Structure Details

```
typedef struct phonemessage_tag {
  DWORD  hDevice;
  DWORD  dwMessageID;
  DWORD_PTR  dwCallbackInstance;
  DWORD_PTR  dwParam1;
  DWORD_PTR  dwParam2;
  DWORD_PTR  dwParam3;
} PHONEMESSAGE, FAR *LPPHONEMESSAGE;
```

Members

hDevice

A handle to a phone device.

dwMessageID

A phone message.

dwCallbackInstance

Instance data that is passed back to the application, which the application specified in phoneInitializeEx. TAPI does not interpret DWORD.

dwParam1

A parameter for the message.

dwParam2

A parameter for the message.

dwParam3

A parameter for the message.

Further Details

For details on the parameter values that are passed in this structure, see "TAPI Phone Messages, on page 286."

PHONESTATUS

The PHONESTATUS structure describes the current status of a phone device. The phoneGetStatus and TSPI_phoneGetStatus functions return this structure.

Device-specific extensions should use the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of this data structure.

Note The dwPhoneFeatures member is available only to applications that open the phone device with an API version of 2.0 or later.

Structure Details

```
typedef struct phonestatus_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwStatusFlags;
  DWORD  dwNumOwners;
  DWORD  dwNumMonitors;
  DWORD  dwRingMode;
  DWORD  dwRingVolume;
  DWORD  dwHandsetHookSwitchMode;
  DWORD  dwHandsetVolume;
  DWORD  dwHandsetGain;
  DWORD  dwSpeakerHookSwitchMode;
  DWORD  dwSpeakerVolume;
  DWORD  dwSpeakerGain;
  DWORD  dwHeadsetHookSwitchMode;
  DWORD  dwHeadsetVolume;
  DWORD  dwHeadsetGain;
  DWORD  dwDisplaySize;
  DWORD  dwDisplayOffset;
  DWORD  dwLampModesSize;
  DWORD  dwLampModesOffset;
  DWORD  dwOwnerNameSize;
  DWORD  dwOwnerNameOffset;
  DWORD  dwDevSpecificSize;
  DWORD  dwDevSpecificOffset;
  DWORD  dwPhoneFeatures;
} PHONESTATUS, FAR *LPPHONESTATUS;
```

Members

dwTotalSize

The total size, in bytes, that is allocated to this data structure.

dwNeededSize

The size, in bytes, for this data structure that is needed to hold all the returned information.

dwUsedSize

The size, in bytes, of the portion of this data structure that contains useful information.

dwStatusFlags

Provides a set of status flags for this phone device. This member uses one of the PHONESTATUSFLAGS_Constants.

dwNumOwners

The number of application modules with owner privilege for the phone.

dwNumMonitors

The number of application modules with monitor privilege for the phone.

dwRingMode

The current ring mode of a phone device.

dwRingVolume
0x8000
dwHandsetHookSwitchMode

The current hookswitch mode of the phone handset. PHONEHOOKSWITCHMODE_UNKNOWN

dwHandsetVolume
0
dwHandsetGain
0
dwSpeakerHookSwitchMode

The current hookswitch mode of the phone speakerphone. PHONEHOOKSWITCHMODE_UNKNOWN

dwSpeakerVolume
0

dwSpeakerGain
0
dwHeadsetHookSwitchMode

The current hookswitch mode of the phone's headset. PHONEHOOKSWITCHMODE_UNKNOWN

dwHeadsetVolume
0
dwHeadsetGain
0
dwDisplaySize
dwDisplayOffset
0
dwLampModesSize
dwLampModesOffset
0
dwOwnerNameSize
dwOwnerNameOffset

The size, in bytes, of the variably sized field that contains the name of the application that is the current owner of the phone device and the offset, in bytes, from the beginning of this data structure. The name is the application name that the application provides when it invokes with phoneInitialize or

phoneInitializeEx. If no application name was supplied, the application's filename is used instead. If the phone currently has no owner, dwOwnerNameSize is zero.

dwDevSpecificSize
dwDevSpecificOffset

Application can send XSI data to phone by using DeviceDataPassThrough device-specific extension. Phone can pass back data to Application. The data is returned as part of this field. The format of the data is as follows:

```
struct PhoneDevSpecificData
{    DWORD m_DeviceDataSize ; // size of device data
    DWORD m_DeviceDataOffset ; // offset from PHONESTATUS
structure
    // this will follow the actual variable length device data.
}
```

dwPhoneFeatures

The application negotiates an extension version > = 0x00020000. The following features are supported:

- PHONEFEATURE_GETDISPLAY
- PHONEFEATURE_GETLAMP
- PHONEFEATURE_GETRING
- PHONEFEATURE_SETDISPLAY
- PHONEFEATURE_SETLAMP

VARSTRING

The VARSTRING structure returns variably sized strings. The line device class and the phone device class both use it.

Note No extensibility exists with VARSTRING.

Structure Details

```
typedef struct varstring_tag {
  DWORD  dwTotalSize;
  DWORD  dwNeededSize;
  DWORD  dwUsedSize;
  DWORD  dwStringFormat;
  DWORD  dwStringSize;
  DWORD  dwStringOffset;
} VARSTRING, FAR *LPVARSTRING;
```

Members

dwTotalSize

The total size, in bytes, that is allocated to this data structure.

dwNeededSize

The size, in bytes, for this data structure that is needed to hold all the returned information.

dwUsedSize

The size, in bytes, of the portion of this data structure that contains useful information.

dwStringFormat

The format of the string. This member uses one of the STRINGFORMAT_Constants.

dwStringSize
dwStringOffset

The size, in bytes, of the variably sized device field that contains the string information and the offset, in bytes, from the beginning of this data structure.

If a string cannot be returned in a variable structure, the dwStringSize and dwStringOffset members get set in one of the following ways:

- dwStringSize and dwStringOffset members both get set to zero.
- dwStringOffset gets set to nonzero and dwStringSize gets set to zero.
- dwStringOffset gets set to nonzero, dwStringSize gets set to 1, and the byte at the given offset gets set to zero.]

Wave Functions

The AVAudio32.dll implements the wave interfaces to the Cisco wave drivers. The system supports all APIs for input and output waveform devices.

Cisco TSP 8.0 includes Cisco Media Driver, a new and innovative way for TAPI-based applications, to provide media interaction. Cisco TSP 8.0(1) includes support for Cisco Media Driver and Cisco Wave Driver. Only one driver is active at any given time. For more information, see Cisco TSP Media Driver, on page 411.

Table 20: Wave Functions

Wave functions
waveInAddBuffer, on page 302
waveInClose, on page 302
waveInGetID, on page 303
waveInGetPosition, on page 303
waveInOpen, on page 304
waveInPrepareHeader, on page 305
waveInReset, on page 306
waveInStart, on page 306

Wave functions
waveInUnprepareHeader, on page 306
waveOutClose, on page 307
waveOutGetDevCaps, on page 307
waveOutGetID, on page 308
waveOutGetPosition, on page 308
waveOutOpen, on page 309
waveOutPrepareHeader, on page 310
waveOutReset, on page 310
waveOutUnprepareHeader, on page 311
waveOutWrite, on page 311

waveInAddBuffer

The waveInAddBuffer function sends an input buffer to the given waveform-audio input device. When the buffer is filled, the application receives notification.

Function Details

```
MMRESULT waveInAddBuffer(
  HWAVEIN hwi,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwi

Handle of the waveform-audio input device.

pwh

Address of a WAVEHDR structure that identifies the buffer.

cbwh

Size, in bytes, of the WAVEHDR structure.

waveInClose

The waveInClose function closes the given waveform-audio input device.

Function Details

```
MMRESULT waveInClose(
  HWAVEIN hwi
);
```

Parameter

hwi

Handle of the waveform-audio input device. If the function succeeds, the handle no longer remains valid after this call.

waveInGetID

The waveInGetID function gets the device identifier for the given waveform-audio input device.

This function gets supported for backward compatibility. New applications can cast a handle of the device rather than retrieving the device identifier.

Function Details

```
MMRESULT waveInGetID(
  HWAVEIN hwi,
  LPUINT puDeviceID
);
```

Parameters

hwi

Handle of the waveform-audio input device.

puDeviceID

Address of a variable to be filled with the device identifier.

waveInGetPosition

The waveInGetPosition function retrieves the current input position of the given waveform-audio input device.

Function Details

```
MMRESULT waveInGetPosition(
  HWAVEIN hwi,
  LPMMTIME pmmt,
  UINT cbmmt
);
```

Parameters

hwi

Handle of the waveform-audio input device.

pmmt

Address of the MMTIME structure.

cbmmt

Size, in bytes, of the MMTIME structure.

waveInOpen

The waveInOpen function opens the given waveform-audio input device for recording.

Function Details

```
MMRESULT waveInOpen(
  LPHWAVEIN phwi,
  UINT uDeviceID,
  LPWAVEFORMATEX pwfx,
  DWORD dwCallback,
  DWORD dwCallbackInstance,
  DWORD fdwOpen
);
```

Parameters

phwi

Address that is filled with a handle that identifies the open waveform-audio input device. Use this handle to identify the device when calling other waveform-audio input functions. This parameter can be NULL if WAVE_FORMAT_QUERY is specified for fdwOpen.HDR structure.

uDeviceID

Identifier of the waveform-audio input device to open. It can be either a device identifier or a handle of an open waveform-audio input device. You can use the following flag instead of a device identifier:

WAVE_MAPPER -The function selects a waveform-audio input device that is capable of recording in the specified format.

pwfx

Address of a WAVEFORMATEX structure that identifies the desired format for recording waveform-audio data. You can free this structure immediately after waveInOpen returns.

Note The formats that the TAPI Wave Driver supports include a 16-bit PCM at 8000 Hz, 8-bit mulaw at 8000 Hz, and 8-bit alaw at 8000 Hz.

dwCallback

Address of a fixed callback function, an event handle, a handle to a window, or the identifier of a thread to be called during waveform-audio recording to process messages that are related to the progress of recording. If no callback function is required, this value can specify zero. For more information on the callback function, see waveInProc in the TAPI API.

dwCallbackInstance

User-instance data that is passed to the callback mechanism. This parameter is not used with the window callback mechanism.

fdwOpen

Flags for opening the device. The following values definitions apply:

- CALLBACK_EVENT -The dwCallback parameter specifies an event handle.
- CALLBACK_FUNCTION -The dwCallback parameter specifies a callback procedure address.
- CALLBACK_NULL -No callback mechanism. This represents the default setting.
- CALLBACK_THREAD -The dwCallback parameter specifies a thread identifier.
- CALLBACK_WINDOW -The dwCallback parameter specifies a window handle.
- WAVE_FORMAT_DIRECT -If this flag is specified, the A driver does not perform conversions on the audio data.
- WAVE_FORMAT_QUERY -The function queries the device to determine whether it supports the given format, but it does not open the device.
- WAVE_MAPPED -The uDeviceID parameter specifies a waveform-audio device to which the wave mapper maps.

waveInPrepareHeader

The waveInPrepareHeader function prepares a buffer for waveform-audio input.

Function Details

```
MMRESULT waveInPrepareHeader(
  HWAVEIN hwi,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwi

Handle of the waveform-audio input device.

pwh

Address of a WAVEHDR structure that identifies the buffer to be prepared.

cbwh

Size, in bytes, of the WAVEHDR structure.

waveInReset

The waveInReset function stops input on the given waveform-audio input device and resets the current position to zero. All pending buffers get marked as done and get returned to the application.

Function Details

```
MMRESULT waveInReset(
  HWAVEIN hwi
);
```

Parameter

hwi

Handle of the waveform-audio input device.

waveInStart

The waveInStart function starts input on the given waveform-audio input device.

Function Details

```
MMRESULT waveInStart(
  HWAVEIN hwi
);
```

Parameter

hwi

Handle of the waveform-audio input device.

waveInUnprepareHeader

The waveInUnprepareHeader function cleans up the preparation that the waveInPrepareHeader function performs. This function must be called after the device driver fills a buffer and returns it to the application. You must call this function before freeing the buffer.

Function Details

```
MMRESULT waveInUnprepareHeader(
  HWAVEIN hwi,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwi

Handle of the waveform-audio input device.

pwh

Address of a WAVEHDR structure that identifies the buffer to be cleaned up.

cbwh

Size, in bytes, of the WAVEHDR structure.

waveOutClose

The waveOutClose function closes the given waveform-audio output device.

Function Details

```
MMRESULT waveOutClose(
  HWAVEOUT hwo
);
```

Parameter

hwo

Handle of the waveform-audio output device. If the function succeeds, the handle no longer remains valid after this call.

waveOutGetDevCaps

The waveOutGetDevCaps function retrieves the capabilities of a given waveform-audio output device.

Function Details

```
MMRESULT waveOutGetDevCaps(
  UINT uDeviceID,
  LPWAVEOUTCAPS pwoc,
  UINT cbwoc
);
```

Parameters

uDeviceID

Identifier of the waveform-audio output device. It can be either a device identifier or a handle of an open waveform-audio output device.

pwoc

Address of a WAVEOUTCAPS structure that is to be filled with information about the capabilities of the device.

cbwoc

Size, in bytes, of the WAVEOUTCAPS structure.

waveOutGetID

The waveOutGetID function retrieves the device identifier for the given waveform-audio output device.

This function gets supported for backward compatibility. New applications can cast a handle of the device rather than retrieving the device identifier.

Function Details

```
MMRESULT waveOutGetID(
  HWAVEOUT hwo,
  LPUINT puDeviceID
);
```

Parameters

hwo

Handle of the waveform-audio output device.

puDeviceID

Address of a variable to be filled with the device identifier.

waveOutGetPosition

The waveOutGetPosition function retrieves the current playback position of the given waveform-audio output device.

Function Details

```
MMRESULT waveOutGetPosition(
  HWAVEOUT hwo,
  LPMMTIME pmmt,
  UINT cbmmt
);
```

Parameters

hwo

Handle of the waveform-audio output device.

pmmt

Address of an MMTIME structure.

cbmmt

Size, in bytes, of the MMTIME structure.

waveOutOpen

The waveOutOpen function opens the given waveform-audio output device for playback.

Function Details

```
MMRESULT waveOutOpen(
  LPHWAVEOUT phwo,
  UINT uDeviceID,
  LPWAVEFORMATEX pwfx,
  DWORD dwCallback,
  DWORD dwCallbackInstance,
  DWORD fdwOpen
);
```

Parameters

phwo

Address that is filled with a handle that identifies the open waveform-audio output device. Use the handle to identify the device when other waveform-audio output functions are called. This parameter might be NULL if the WAVE_FORMAT_QUERY flag is specified for fdwOpen.

uDeviceID

Identifier of the waveform-audio output device to open. It can be either a device identifier or a handle of an open waveform-audio input device. You can use the following flag instead of a device identifier:

WAVE_MAPPER -The function selects a waveform-audio output device that is capable of playing the given format.

pwfx

Address of a WAVEFORMATEX structure that identifies the format of the waveform-audio data to be sent to the device. You can free this structure immediately after passing it to waveOutOpen.

Note The formats that the TAPI Wave Driver supports include 16-bit PCM at 8000 Hz, 8-bit mulaw at 8000 Hz, and 8-bit alaw at 8000 Hz.

dwCallback

Address of a fixed callback function, an event handle, a handle to a window, or the identifier of a thread to be called during waveform-audio playback to process messages that are related to the progress of the playback. If no callback function is required, this value can specify zero. For more information on the callback function, see waveOutProc in the TAPI API.

dwCallbackInstance

User-instance data that is passed to the callback mechanism. This parameter is not used with the window callback mechanism.

fdwOpen

Flags for opening the device. The following value definitions apply:

- CALLBACK_EVENT -The dwCallback parameter represents an event handle.

- CALLBACK_FUNCTION -The dwCallback parameter specifies a callback procedure address.
- CALLBACK_NULL -No callback mechanism. This value specifies the default setting.
- CALLBACK_THREAD -The dwCallback parameter represents a thread identifier.
- CALLBACK_WINDOW -The dwCallback parameter specifies a window handle.
- WAVE_ALLOWSYNC -If this flag is specified, a synchronous waveform-audio device can be opened. If this flag is not specified while a synchronous driver is opened, the device will fail to open.
- WAVE_FORMAT_DIRECT -If this flag is specified, the ACM driver does not perform conversions on the audio data.
- WAVE_FORMAT_QUERY -If this flag is specified, waveOutOpen queries the device to determine whether it supports the given format, but the device does not actually open.
- WAVE_MAPPED -If this flag is specified, the uDeviceID parameter specifies a waveform-audio device to which the wave mapper maps.

waveOutPrepareHeader

The waveOutPrepareHeader function prepares a waveform-audio data block for playback.

Function Details

```
MMRESULT waveOutPrepareHeader(
  HWAVEOUT hwo,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwo

Handle of the waveform-audio output device.

pwh

Address of a WAVEHDR structure that identifies the data block to be prepared.

cbwh

Size, in bytes, of the WAVEHDR structure.

waveOutReset

The waveOutReset function stops playback on the given waveform-audio output device and resets the current position to zero. All pending playback buffers get marked as done and get returned to the application.

Function Details

```
MMRESULT waveOutReset(
  HWAVEOUT hwo
);
```

Parameter

hwo

Handle of the waveform-audio output device.

waveOutUnprepareHeader

The waveOutUnprepareHeader function cleans up the preparation that the waveOUtPrepareHeader function performs. Ensure this function is called after the device driver is finished with a data block. You must call this function before freeing the buffer.

Function Details

```
MMRESULT waveOutUnprepareHeader(
  HWAVEOUT hwo,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwo

Handle of the waveform-audio output device.

pwh

Address of a WAVEHDR structure that identifies the data block to be cleaned up.

cbwh

Size, in bytes, of the WAVEHDR structure.

waveOutWrite

The waveOutWrite function sends a data block to the given waveform-audio output device.

Function Details

```
MMRESULT waveOutWrite(
  HWAVEOUT hwo,
  LPWAVEHDR pwh,
  UINT cbwh
);
```

Parameters

hwo

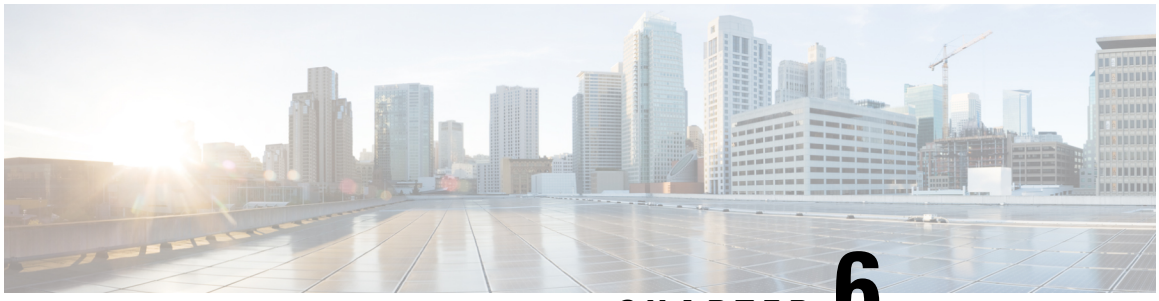
Handle of the waveform-audio output device.

pwh

Address of a WAVEHDR structure that contains information about the data block.

cbwh

Size, in bytes, of the WAVEHDR structure.



CHAPTER 6

Cisco Device-Specific Extensions

This chapter describes the Cisco device-specific TAPI extensions. CiscoLineDevSpecific and the CCiscoPhoneDevSpecific class represent the parent class. This chapter describes how to invoke the Cisco device-specific TAPI extensions with the lineDevSpecific function. It also describes a set of classes that you can use when you call phoneDevSpecific. It contains the following sections:

Cisco Line Device Specific Extensions

The following table lists and describes the subclasses of Cisco Line Device-Specific Extensions. This section contains all of the extensions in the table and descriptions of the following data structures:

Cisco functions	Synopsis
CCiscoLineDevSpecific, on page 339	The CCiscoLineDevSpecific class specifies the parent class to the following classes.
Message Waiting, on page 342	The CCiscoLineDevSpecificMsgWaiting class turns the message waiting lamp on or off for the line that the hLine parameter specifies.
Message Waiting Dirn, on page 343	The CCiscoLineDevSpecificMsgWaiting class turns the message waiting lamp on or off for the line that a parameter specifies and remains independent of the hLine parameter.

Cisco functions	Synopsis
Message Summary, on page 344	The CCiscoLineDevSpecificSetMsgSummary class turns the message waiting lamp on or off, as well as provides voice and fax message counts for the line specified by the hLine parameter.
Message Summary Dirn, on page 346	The CCiscoLineDevSpecificSetMsgSummaryDirn class turns the message waiting lamp on or off and provides voice and fax message counts for the line specified by a parameter and is independent of the hLine parameter.
Audio Stream Control, on page 347	The CCiscoLineDevSpecificUserControlRTPStream class controls the audio stream for a line.
Set Status Messages, on page 349	The CCiscoLineDevSpecificSetStatusMsgs class controls the reporting of certain line device specific messages for a line.
Swap-Hold/SetupTransfer, on page 352	Cisco Unified TSP 4.0 and later do not support this function. The CCiscoLineDevSpecificSwapHoldSetupTransfer class performs a setupTransfer between a call that is in CONNECTED state and a call that is in ONHOLD state. This function will change the state of the connected call to ONHOLDPENDTRANSFER state and the ONHOLD call to CONNECTED state. This action will then allow a completeTransfer to be performed on the two calls.
Redirect Reset Original Called ID, on page 353	The CCiscoLineDevSpecificRedirectResetOrigCalled class gets used to redirect a call to another party while resetting the original called ID of the call to the destination of the redirect.
Port Registration per Call, on page 353	The CciscoLineDevSpecificPortRegistrationPerCall class gets used to register a CTI port or route point for the Dynamic Port Registration feature, which allows applications to specify the IP address and UDP port number on a call-by-call basis.
Setting RTP Parameters for Call, on page 356	The CciscoLineDevSpecificSetRTPParamsForCall class sets the IP address and UDP port number for the specified call.
Redirect Set Original Called ID, on page 356	The CCiscoLineDevSpecificRedirectSetOrigCalled class to redirect a call to another party while setting the original called ID of the call to any other party.
Join, on page 357	The CciscoLineDevSpecificJoin class joins two or more calls into one conference call.
Set User SRTP Algorithm IDs, on page 358	The CciscoLineDevSpecificUserSetSRTPAlgorithmID class allows the application to set SRTP algorithm IDs. You should use this class after lineopen and before CCiscoLineDevSpecificSetRTPParamsForCall or CCiscoLineDevSpecificUserControlRTPStream

Cisco functions	Synopsis
Explicit Acquire, on page 360	The CciscoLineDevSpecificAcquire class explicitly acquires any CTI Controllable device in the Cisco Unified Communications Manager system, which needs to be opened in Super Provider mode.
Explicit De-Acquire, on page 360	The CciscoLineDevSpecificDeacquire class explicitly de-acquires any CTI controllable device in the Cisco Unified Communications Manager system.
Redirect FAC CMC, on page 361	The CCiscoLineDevSpecificRedirectFACCMC class redirects a call to another party while including a FAC, CMC, or both.
Blind Transfer FAC CMC, on page 362	The CCiscoLineDevSpecificBlindTransferFACCMC class blind transfers a call to another party while including a FAC, CMC, or both.
CTI Port Third Party Monitor, on page 363	The CCiscoLineDevSpecificCTIPortThirdPartyMonitor class opens a CTI port in third-party mode.
Send Line Open, on page 364	The CciscoLineDevSpecificSendLineOpen class triggers actual line open from TSP side. Use this for delayed open mechanism.
Set Intercom SpeedDial, on page 365	The CciscoLineSetIntercomSpeeddial class allows the application to set or reset SpeedDial/Label on an intercom line.
Intercom Talk Back, on page 366	The CciscoLineIntercomTalkback class allows the application to initiate talk back on an incoming Intercom call on an Intercom line.
Redirect with Feature Priority, on page 367	The CciscoLineRedirectWithFeaturePriority class enables the application to redirect calls with specified priority.
Start Call Monitoring, on page 367	The CCiscoLineDevSpecificStartCallMonitoring class allows applications to send a start monitoring request for the active call on a line.
Start Call Recording, on page 369	The CCiscoLineDevSpecificStartCallRecording allows the application to send a recording request for the active call on that line.
StopCall Recording, on page 370	The CCiscoLineDevSpecificStopCallRecording allows the application to stop recording a call on that line.
Set IPv6 Address and Mode, on page 371	The CciscoLineDevSpecificSetIPv6AddressAndMode enables the application to set the IPv6 address and addressing mode during registration.
Set RTP Parameters for IPv6 Calls, on page 372	The CciscoLineDevSpecificSetRTPParamsForCallIPv6 class sets the RTP parameters for calls for which you must specify IPv6 address.

Cisco functions	Synopsis
Direct Transfer, on page 373	The CciscoLineDevSpecificDirectTransfer class transfers calls across lines or on the same line.
RegisterCallPickUpGroupForNotification, on page 374	The CciscoLineDevSpecificRegisterCallPickupGroupForNotification class is used to register the call Pickup Group for notification on calls for Pickup.
UnRegisterCallPickUpGroupForNotification, on page 375	The CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification class is used to unregister the call Pickup Group for notification on calls for Pickup.
CallPickUpRequest, on page 375	This feature allows to invoke the pickup, group-pickup, other-pickup, and directed pickup feature from the application. Apart from providing API to invoke feature, application will have capability to register Call pickup group for alert notification, whenever a call is available for pickup. The CciscoLineDevSpecificCallPickupRequest class is used to Pickup the call from the PickGroup.
Start Send Media to BIB, on page 376	The CCiscoLineDevSpecificStartSendMediaToBIBRequest class allows the application to initiate agent greeting to the customer call.
Stop Send Media to BIB, on page 377	The CCiscoLineDevSpecificStopSendMediaToBIBRequest class allows the application to stop agent greeting that is playing on the agent-to-customer call.
Agent Zip Tone, on page 378	The CciscoLineDevSpecificEnableFeatureSupport class allows the application to initiate Zip tone on the Agent Call.
Enable Feature, on page 379	The CciscoLineDevSpecificEnableFeatureSupport class allows the application to enhance or update feature support.
Add Remote Destination, on page 382	The CciscoLineDevSpecificAddRemoteDestination class is used to add new Remote Destination to CTI Remote Device.
Remove Remote Destination, on page 383	The CciscoLineDevSpecificRemoveRemoteDestination class is used to remove Remote Destination from List of Remote Destinations of CTI Remote Device.
Update Remote Destination, on page 384	The CciscoLineDevSpecificUpdateRemoteDestination class is used to update Remote Destination information on a CTI Remote Device.
lineHold Enhancement, on page 385	CciscoLineDevSpecificHoldEx class is used to put call on hold and specify media content that should be played while call is on hold.

Cisco functions	Synopsis
Transfer with media	The CciscoSetupTransferWithoutMedia class allows the application to transfer a call that does not have media setup.

LINEDEVCAPS

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINEDEVCAPS data structure for those extensions. The Cisco_LineDevCaps_Ext structure in the CiscoLineDevSpecificMsg.h header file defines the DevSpecific area layout. Cisco TSP organizes the data in that structure based on the extension version in which the data was introduced:

```
//   LINEDEVCAPS Dev Specific extention   //
typedef struct Cisco_LineDevCaps_Ext
{
    Cisco_LineDevCaps_Ext00030000  ext30;
    Cisco_LineDevCaps_Ext00060000  ext60;
    Cisco_LineDevCaps_Ext00070000  ext70;
    Cisco_LineDevCaps_Ext00080000  ext80;
    Cisco_LineDevCaps_Ext00090000  ext90;
    Cisco_LineDevCaps_Ext00090001  ext91;
    Cisco_LineDevCaps_Ext000A0000  extA0;
    Cisco_LineDevCaps_Ext000C0000  extC0;
    Cisco_LineDevCaps_Ext000D0000  extD0;
    Cisco_LineDevCaps_Ext000E0000  extE0;
```

```
} CISCO_LINEDEVCAPS_EXT;
```

For a specific line device, the extension area will include a portion of this structure starting from the beginning and up to the extension version that an application negotiated.

The individual extension version substructure definitions follow:

```
//    LINEDEVCAPS 00030000  extention    //
typedef struct Cisco_LineDevCaps_Ext00030000
{
    DWORD dwLineTypeFlags;
} CISCO_LINEDEVCAPS_EXT00030000;

//    LINEDEVCAPS 00060000  extention    //
typedef struct Cisco_LineDevCaps_Ext00060000
{
DWORD dwLocale;
} CISCO_LINEDEVCAPS_EXT00060000;

//    LINEDEVCAPS 00070000  extention    //
typedef struct Cisco_LineDevCaps_Ext00070000
{
    DWORD dwPartitionOffset;
    DWORD dwPartitionSize;
} CISCO_LINEDEVCAPS_EXT00070000;

//    LINEDEVCAPS  00080000  extention    //
typedef struct Cisco_LineDevCaps_Ext00080000
{
```

```
    DWORD                   dwLineDevCaps_DevSpecificFlags;        //
 LINEFEATURE_DEVSPECIFIC
    DWORD                   dwLineDevCaps_DevSpecificFeatureFlags; //
 LINEFEATURE_DEVSPECIFICFEAT
    RECORD_TYPE_INFO        recordTypeInfo;
    INTERCOM_SPEEDDIAL_INFO intercomSpeedDialInfo;
} CISCO_LINEDEVCAPS_EXT00080000;

//    LINEDEVCAPS  00090000  extention    //
//    -------------------------------
typedef struct Cisco_LineDevCaps_Ext00090000
{
    IpAddressingMode      dwLineDevCapsIPAddressingMode;        //
 LINEFEATURE_DEVSPECIFIC
} CISCO_LINEDEVCAPS_EXT00090000;

// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = = = = = = = = =
//            Cisco Extention 00090001
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 = = = = = = = = = =
//    LINEDEVCAPS  00090001  extention    //
//    -------------------------------
typedef struct Cisco_LineDevCaps_Ext00090001
{
    DWORD   MaxCalls ;
    DWORD   BusyTrigger ;
    DWORD   LineInstanceNumber  ;
    DWORD   LineLabelASCIIOffset  ;
    DWORD   LineLabelASCIISize  ;
    DWORD   LineLabelUnicodeOffset  ;
    DWORD   LineLabelUnicodeSize  ;
DWORD   VoiceMailPilotDNOffset  ;
DWORD   VoiceMailPilotDNSize  ;
DWORD   RegisteredIPAddressMode;// IpAddressingMode
    DWORD   RegisteredIPv4Address   ;
    DWORD   RegisteredIPv6AddressOffset;
DWORD   RegisteredIPv6AddressSize;
DWORD   ApplicationFeatureFlagBitMap;// CiscoFeatureInformation
DWORD   DeviceFeatureFlagBitMap; // CiscoFeatureInformation
} CISCO_LINEDEVCAPS_EXT00090001;

typedef struct Cisco_LineDevCaps_Ext000A0000
{
    DWORD dwPickUpGroupDNOffset;
    DWORD dwPickUpGroupDNSize;
    DWORD dwPickUpGroupPartitionOffset;
    DWORD dwPickUpGroupPartitionSize;
} CISCO_LINEDEVCAPS_EXT000A0000;

typedefstruct Cisco_LineDevCaps_Ext000C0000
{
    DWORD DeviceProtocolType;
    DWORD RemoteDestinationOffset;
    DWORD RemoteDestinationSize;
    DWORD RemoteDestinationCount;
    DWORD RemoteDestinationElementFixedSize;
    DWORD IsMyAppLastToSetActiveRD;
} CISCO_LINEDEVCAPS_EXT000C0000;
typedef struct Cisco_LineDevCaps_Ext000D0000
{
DWORD DeviceMultiMediaCapabilityBitMask;
DWORD DeviceMultiMediaCapabilityOffset;
DWORD DeviceMultiMediaCapabilitySize;
```

```
DWORD DeviceMultiMediaCapabilityCount;
DWORD DeviceMultiMediaCapabilityElementFixedSize;
DWORD ClusterIDSize;
DWORD ClusterIDOffset;
} CISCO_LINEDEVCAPS_EXT000D0000;
```

See the CiscoLineDevSpecificMsg.h header file for additional information on the DevSpecific structure layout and data.

CISCO_LINEDEVCAPS_EXT000D000 structure contains following information:

Table 21: CISCO_LINEDEVCAPS_EXT000D000 Structure

Fields	Description
DWORD DeviceMultiMediaCapabilityBitMask	Bitmask field indicates which fields in MultiMediaCapability structure Info are valid
DWORD DeviceMultiMediaCapabilityOffset	Offset pointing to the DeviceMultiMediaCapability structure Information
DWORD DeviceMultiMediaCapabilitySize	Size of the DeviceMultiMediaCapability Information
DWORD DeviceMultiMediaCapabilityCount	Count of MultiMediaCapaility Info available
DWORD DeviceMultiMediaCapabilityElementFixedSize	Size of MultiMediaCapability Structure
DWORD ClusterIDSize DWORD ClusterIDOffset	Offset/Size of the name of the cluster ID where the line is located

MultiMediacapability Information

CiscoDeviceMultiMediaCapInfoBitMask - Bit mask indicates which fields in MultiMediaCapability Structure Exposed are valid and can be used by Applications. Following is the Enum Definition which is used to update this bitmask field.

```
enum CiscoDeviceMultiMediaCapInfoBitMask
{
    CiscoDeviceMultiMediaCapability_None                    = 0x00000000,
    CiscoDeviceMultiMediaCapability_VideoCapability         = 0x00000001,
    CiscoDeviceMultiMediaCapability_TelepresenceInfo        = 0x00000002,
    CiscoDeviceMultiMediaCapability_ScreenCount             = 0x00000004
};
```

Device MultiMedia Capability of the Device is exposed as a structure DeviceMultiMediaCapability in the DevSpecific part. This structure contains the fields deviceVideoCapability, telepresenceInfo and screenCount.

```
typedef struct DeviceMultiMediaCapability
{
    DWORD deviceVideoCapability;
    DWORD telepresenceInfo;
    DWORD screenCount;
} DeviceMultiMediaCapability;
```

Data fields	Value
DeviceVideoCapability	This field contains the type value defined in the following enumeration. [CiscoDeviceVideoCapabilityInfo]
TelepresenceInfo	This field indicates if Telepresence interop is supported by the device, defined in the following enumeration. [CiscoDeviceTelepresenceInfo]
ScreenCount	This field indicates the number of screens present on the device.

```
enum CiscoLineDeviceVideoCapabilityInfo
{
    CiscoLineDeviceVideoCapability_None =          0x00000000,
    CiscoLineDeviceVideoCapability_Enabled =       0x00000001,
};

enum CiscoDeviceTelepresenceInfo
{
    CiscoDeviceTelepresence_None =          0x00000000,
    CiscoDeviceTelepresence_Enabled =       0x00000001,
};
```

LINECALLINFO

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINECALLINFO data structure for those extensions. The Cisco_LineCallInfo_Ext structure in the CiscoLineDevSpecificMsg.h header file defines DevSpecific area layout. Cisco TSP organizes the data in the structure, that is based on the extension version, in which the data is introduced:

```
//   LINECALLINFO Dev Specific extention   //
typedef struct Cisco_LineCallInfo_Ext
{
    Cisco_LineCallInfo_Ext00060000  ext60;
    Cisco_LineCallInfo_Ext00070000  ext70;
    Cisco_LineCallInfo_Ext00080000  ext80;
    Cisco_LineCallInfo_Ext00080001  ext81;
    Cisco_LineCallInfo_Ext00090000  ext90;
    Cisco_LineCallInfo_Ext00090000  ext91;
    Cisco_LineCallInfo_Ext000A0000  extA0;
    Cisco_LineCallInfo_Ext000D0000  extD0;
    ;
```

```
} CISCO_LINECALLINFO_EXT;
```

For a specific line device, the extension area includes, a portion of the structure from the beginning to the extension version that an application negotiated.

The definitions for individual extension version substructure are as follows:

```
//    LINECALLINFO 00060000  extention    //
typedef struct Cisco_LineCallInfo_Ext00060000
{
    TSP_UNICODE_PARTY_NAMES  unicodePartyNames;
} CISCO_LINECALLINFO_EXT00060000;
```

```
//    LINECALLINFO 00070000  extention    //
typedef struct Cisco_LineCallInfo_Ext00070000
{
    DWORD SRTPKeyInfoStructureOffset;   // offset from base of LINECALLINFO
    DWORD SRTPKeyInfoStructureSize;     // includes variable length data total
size
    DWORD SRTPKeyInfoStructureElementCount;
    DWORD SRTPKeyInfoStructureElementFixedSize;
    DWORD DSCPInformationOffset;        // offset from base of LINECALLINFO
    DWORD DSCPInformationSize;          // fixed size of the DSCPInformation
structure
    DWORD DSCPInformationElementCount;
    DWORD DSCPInformationElementFixedSize;
    DWORD CallPartitionInfoOffset;      // offset from base of LINECALLINFO
    DWORD CallPartitionInfoSize;        // fixed size of the
CallPartitionInformation
 structure
    DWORD CallPartitionInfoElementCount;
    DWORD CallPartitionInfoElementFixedSize;
    DWORD ExtendedCallInfoOffset;       //  = = = > ExtendedCallInfo { }
    DWORD ExtendedCallInfoSize;         //
    DWORD ExtendedCallInfoElementCount; //
    DWORD ExtendedCallInfoElementSize;  //
} CISCO_LINECALLINFO_EXT00070000;

//    LINECALLINFO 00080000  extention    //
//    ---------------------------------
typedef struct Cisco_LineCallInfo_Ext00080000
{
    DWORD CallSecurityStatusOffset;
    DWORD CallSecurityStatusSize;
    DWORD CallSecurityStatusElementCount;
    DWORD CallSecurityStatusElementFixedSize;
    DWORD CCMCallIDInfoOffset;
    DWORD CCMCallIDInfoSize;
    DWORD CCMCallIDInfoElementCount;
    DWORD CCMCallIDInfoElementFixedSize;
    DWORD CallAttrtibuteInfoOffset;
    DWORD CallAttrtibuteInfoSize;
    DWORD CallAttrtibuteInfoElementCount;
    DWORD CallAttrtibuteInfoElementFixedSize;
    DWORD TSPIntercomSideInfo;
    DWORD CallingPartyIpAddr;
} CISCO_LINECALLINFO_EXT00080000;

//    LINECALLINFO 00080001  extension    //
//    ---------------------------------
typedef struct Cisco_LineCallInfo_Ext00080001
{
    DWORD CPNInfoOffset;         //array of structure of CPNInfo structure
    DWORD CPNInfoSize;
    DWORD CPNInfoElementCount;
    DWORD CPNInfoElementFixedSize;

} CISCO_LINECALLINFO_EXT00080001;
//    LINECALLINFO 00090000  extention    //
//    ---------------------------------
typedef struct Cisco_LineCallInfo_Ext00090000
{
    DWORD IPv6InfoOffset;
    DWORD IPv6InfoSize;
    DWORD IPv6InfoElementCount;
    DWORD IPv6InfoElementFixedSize;
    DWORD FarEndIPAddressingMode;
```

```
}CISCO_LINECALLINFO_EXT00090000;

//    LINECALLINFO 000A0000  extention    //
//    --------------------------------
typedef struct Cisco_LineCallInfo_Ext000A0000
{
    DWORD CallAttributeBitMask;
    DWORD UniqueCallRefIDInfoOffset;
    DWORD UniqueCallRefIDInfoSize;
    DWORD UniqueCallRefIDInfoElementCount;
    DWORD UniqueCallRefIDElementFixedSize;
    //HuntList
    DWORD HuntPilotInfoOffset; //point to HuntPoiltInfo
    DWORD HuntPilotInfoSize;
    DWORD HuntPilotInfoCount;
    DWORD HuntPilotInfoElementFixedSize;
    DWORD  GlobalCallID;
    DWORD  CallManagerID;
} CISCO_LINECALLINFO_EXT000A0000;
```

```
typedef struct Cisco_LineCallInfo_Ext000D0000
{
    DWORD CallingPartyMultiMediaCapBitMask; //refer to
CiscoDeviceMultiMediaCapInfoBitMask
    DWORD CalledPartyMultiMediaCapBitMask; //refer to
CiscoDeviceMultiMediaCapInfoBitMask
    DWORD CallingPartyMultiMediaCapInfoOffset; //refer to
DeviceCallMultiMediaCapInfo
    DWORD CallingPartyMultiMediaCapInfoSize;
    DWORD CallingPartyMultiMediaCapInfoCount;
    DWORD CallingPartyMultiMediaCapInfoElementFixedSize;
    DWORD CalledPartyMultiMediaCapInfoOffset; //refer to
DeviceCallMultiMediaCapInfo
    DWORD CalledPartyMultiMediaCapInfoSize;
    DWORD CalledPartyMultiMediaCapInfoCount;
    DWORD CalledPartyMultiMediaCapInfoElementFixedSize;
    DWORD MultiMediaStreamsInfoOffset; //refer to VideoStreamInfo
    DWORD MultiMediaStreamsInfoSize;
    DWORD MultiMediaStreamsInfoCount;
    DWORD MultiMediaStreamsInfoElementFixedSize;
    DWORD RecordingAttributeInfo_ExtD0_Offset;
    DWORD RecordingAttributeInfo_ExtD0_Size;
    DWORD RecordingAttributeInfo_ExtD0_Count;
    DWORD RecordingAttributeInfo_ExtD0_ElementFixedSize;
} CISCO_LINECALLINFO_EXT000D0000;
```

Calling and Called MultiMediaCapability Information

The video capability of the calling party and the called party is exposed as a structure DeviceCallMultiMediaCapInfo in the DevSpecific part. The structure contains the following fields:

- VideoCapStatus,
- TelepresenceInfo, and
- ScreenCount

```
typedef struct DeviceCallMultiMediaCapInfo
{
DWORD VideoCapStatus;
DWORD TelepresenceInfo;
```

```
DWORD ScreenCount;
} DeviceCallMultiMediaCapInfo;
```

Data fields	Value
DeviceVideoCapability	Contains the value that is defined in the following enumeration [CiscoDeviceVideoCapabilityInfo].
TelepresenceInfo	Indicates if Telepresence is enabled on the device, which is defined in the following enumeration [CiscoDeviceTelepresenceInfo].
ScreenCount	Indicates the number of screens present on the device.
CallingPartyMultiMediaCapInfoBitMask	Indicates which fields of DeviceCallMultiMediaCapInfo structure have valid information [CiscoDeviceMultiMediaCapInfoBitMask].
CalledPartyMultiMediaCapInfoBitMask	Indicates which fields of DeviceCallMultiMediaCapInfo structure have valid information [CiscoDeviceMultiMediaCapInfoBitMask].

```
enum CiscoDeviceVideoCapabilityInfo
{
CiscoDeviceVideoCapability_None = 0x00000000,
CiscoDeviceVideoCapability_Enabled = 0x00000001,
};
enum CiscoDeviceTelepresenceInfo
{
CiscoDeviceTelepresence_None = 0x00000000,
CiscoDeviceTelepresence_Enabled = 0x00000001,
};
enum CiscoDeviceMultiMediaCapInfoBitMask
{
CiscoDeviceMultiMediaCapability_None = 0x00000000,
CiscoDeviceMultiMediaCapability_VideoCapability = 0x00000001,
CiscoDeviceMultiMediaCapability_TelepresenceInfo = 0x00000002,
CiscoDeviceMultiMediaCapability_ScreenCount = 0x00000004
};
```

MultiMediaStream Information

When the call arrives on an opened line, the TSP sends the LINE_CALLDEVSPECIFIC event to the application with Multimedia Stream information.

The application then sends a query to the LINECALLINFO to get a detailed Multimedia Stream information. The information is exposed as a part of the VideoStreamInfo structure in the DevSpecific part of the LineCallInfo.

The structure contains the following data.

```
typedef struct VideoStreamInfo
{
    DWORD StreamId;
    DWORD CompressionType; // MEDIAPAYLOAD
```

```
    DWORD BitRate;
    DWORD MediaMode;
    DWORD bKeyInfoPresent;
    //ipv6
    DWORD RxRTPDestinationV6Offset;
    DWORD RxRTPDestinationV6Size;
    DWORD RxRTPDestinationV4;
    DWORD RxIpAddrMode;
    DWORD TxRTPDestinationV6Offset;
    DWORD TxRTPDestinationV6Size;
    DWORD TxRTPDestinationV4;
    DWORD TxIpAddrMode;
    MultiMediaEncryptionKeyInfo MediaEncryptionKeyInfo;
} VideoStreamInfo;
```

Data fields	Value
StreamId	Indicates the index of the MultiMedia stream.
CompressionType	Indicates the compression type of the video stream.
BitRate	Indicates the bit rate of the video stream.
MediaMode	Indicates the media mode of the video stream.
PacketSize	Indicates the packet size of the video stream.
bKeyInfoPresent	Indicates whether Key Information is present.
RxRTPDestinationV6Offset	Contains the value in bytes from the beginning of LINECALLINFO structure.
RxRTPDestinationV6Size	Contains the value in bytes of the variably sized Reception RTP destination IPv6 information.
RxRTPDestinationV4	Indicates the IPv4 address of the video stream.
RxIpAddrMode	Specifies the reception IP addressing mode.
TxRTPDestinationV6Offset	Contains the value in bytes from the beginning of LINECALLINFO structure.
TxRTPDestinationV6Size	Contains the value in bytes of the variably sized Transmission RTP destination IPv6 information.
TxRTPDestinationV4	Indicates the IPv4 address of the video stream.
TxIpAddrMode	Specifies the transmission IP addressing mode.
MediaEncryptionKeyInfo	Contains the value in bytes from the beginning of LINECALLINFO structure. Contains the value in bytes of the variably sized Multimedia Encryption Key information.

Cisco TSP reports a detailed multimedia Encryption Key Information to the applications as a part of the structure CiscoTsp_MultiMediaEncryptionKeyInfo, if there is secure connection to CTIManager. The application user is authorized to receive multimedia Encryption Key Information.

The multimedia Encryption Key information that is provided by Cisco TSP includes TxKeylen, RxKeylen, Txkey, RxKey, TxSalt, RxSalt, AlgorithmID, TxIsMKIPresent, RxIsMKIPresent, and SecurityIndicator.

The administrator must configure TLS Enabled and SRTP Enabled flags on CallManager Admin User pages to receive the key materials. TLS link must be established between TSP and CTIManager.

```
typedef struct CiscoTsp_MultiMediaEncryptionKeyInfo
{
DWORD AlgorithmID;
DWORD TxKeyOffset;
DWORD TxKeySize;
DWORD RxKeyOffset;
DWORD RxKeySize;
DWORD TxSaltOffset;
DWORD TxSaltSize;
DWORD RxSaltOffset;
DWORD RxSaltSize;
DWORD TxIsMKIPresent;
DWORD RxIsMKIPresent;
DWORD SecurityIndicator;
} CiscoTsp_MultiMediaEncryptionKeyInfo;
```

AlgorithmID	Specifies the negotiated algorithm id.
TxKeyOffset	Contains the value in bytes from the beginning of LINECALLINFO structure.
TxKeySize	Contains the value in bytes of the variably sized Transmission Key information.
RxKeyOffset	Contains the value in bytes from the beginning of LINECALLINFO structure.
RxKeySize	Contains the value in bytes of the variably sized Reception Key information.
TxSaltOffset	Contains the value in bytes from the beginning of LINECALLINFO structure.
TxSaltSize	Contains the value in bytes of the variably sized Transmission Salt information.
RxSaltOffset	Contains the value in bytes from the beginning of LINECALLINFO structure.
RxSaltSize	Contains the value in bytes of the variably sized Reception Salt information.
TxIsMKIPresent	Indicates whether Transmission MKI is present.
RxIsMKIPresent	Indicates whether Reception MKI is present.
SecurityIndicator	Specifies the security indicator.

See the CiscoLineDevSpecificMsg.h header file for additional information on the DevSpecific structure layout and data.

Details

The TSP_Unicode_Party_names structure and SRTP information structure describe the device-specific extensions that the Cisco Unified TSP made to the LINECALLINFO structure. DSCPValueForAudioCalls will contain the DSCP value that CTI sent in the StartTransmissionEvent.

ExtendedCallInfo structure has extra call information. For Cisco Unified Communications Manager Release 7.0(1), the ExtendedCallReason field belongs to the ExtendedCallInfo structure.

CallAttributeInfo contains the information about attributeType (Monitoring, Monitored, Recorder,securityStatus) and PartyInfo (Dn,Partition,DeviceName)

CCMCallID contains CCM Call identifier value.

CallingPartyIPAddress contains the IP address of the calling party if the calling party device supports it.

CallSecurityStatus structure contains the overall security status of the call for two-party call as well as conference call.

```
DWORD TapiCallerPartyUnicodeNameOffset;
DWORD TapiCallerPartyUnicodeNameSize;
DWORDTapiCallerPartyLocale;

DWORD TapiCalledPartyUnicodeNameOffset;
DWORD TapiCalledPartyUnicodeNameSize;
DWORDTapiCalledPartyLocale;

DWORD TapiConnectedPartyUnicodeNameOffset;
DWORD TapiConnectedPartyUnicodeNameSize;
DWORDTapiConnectedPartyLocale;

DWORD TapiRedirectionPartyUnicodeNameOffset;
DWORD TapiRedirectionPartyUnicodeNameSize;
DWORDTapiRedirectionPartyLocale;

DWORD TapiRedirectingPartyUnicodeNameOffset;
DWORD TapiRedirectingPartyUnicodeNameSize;
DWORDTapiRedirectingPartyLocale;

DWORD SRTPKeyInfoStructureOffset; // offset from base of LINECALLINFO
DWORD SRTPKeyInfoStructureSize;// includes variable length data total size
DWORD SRTPKeyInfoStructureElementCount;
DWORD SRTPKeyInfoStructureElementFixedSize;
DWORD DSCPValueInformationOffset;
DWORD DSCPValueInformationSize;
DWORD DSCPValueInformationElementCount;
DWORD DSCPValueInformationElementFixedSize;
DWORD PartitionInformationOffset; // offset from base of LINECALLINFO
DWORD PartitionInformationSize;  // includes variable length data total size
DWORD PartitionInformationElementCount;
DWORD PartitionInformationElementFixedSize;
DWORD ExtendedCallInfoOffset;
DWORD ExtendedCallInfoSize;
DWORD ExtendedCallInfoElementCount;
DWORD ExtendedCallInfoElementSize;
DWORD CallAttrtibuteInfoOffset;
DWORD CallAttrtibuteInfoSize;
DWORD CallAttrtibuteInfoElementCount;
```

```
DWORD CallAttrtibuteInfoElementSize;
DWORD CallingPartyIPAddress;
DWORD CCMCallIDInfoOffset;
DWORD CCMCallIDInfoSize;
DWORD CCMCallIDInfoElementCount;
DWORD CCMCallIDInfoElementFixedSize;
DWORD CallSecurityStatusOffset;
DWORD CallSecurityStatusSize;
DWORD CallSecurityStatusElementCount;
DWORD CallSecurityStatusElementFixedSize;
DWORD IsChaperoneCall;
DWORD UniqueCallRefIDInfoOffset,
DWORD UniqueCallRefIDInfoSize;
DWORD CallAttributeBitMask;

typedef struct SRTPKeyInfoStructure
{
SRTPKeyInformation TransmissionSRTPInfo;
SRTPKeyInformation ReceptionSRTPInfo;
} SRTPKeyInfoStructure;

typedef struct SRTPKeyInformation
{
DWORDIsSRTPDataAvailable;
DWORDSecureMediaIndicator;// CiscoSecurityIndicator
DWORDMasterKeyOffset;
DWORDMasterKeySize;
DWORDMasterSaltOffset;
DWORDMasterSaltSize;
DWORDAlgorithmID;// CiscoSRTPAlgorithmIDs
DWORDIsMKIPresent;
DWORDKeyDerivationRate;
} SRTPKeyInformation;
enum CiscoSRTPAlgorithmIDs
{
SRTP_NO_ENCRYPTION=0,
SRTP_AES_128_COUNTER=1
};

enum CiscoSecurityIndicator
{
    SRTP_MEDIA_ENCRYPT_KEYS_AVAILABLE,
    SRTP_MEDIA_ENCRYPT_USER_NOT_AUTH,
    SRTP_MEDIA_ENCRYPT_KEYS_UNAVAILABLE,
    SRTP_MEDIA_NOT_ENCRYPTED
};
```

If isSRTPInfoavailable is set to False, rest of the information from SRTPKeyInformation must be ignored.

If MasterKeySize or MasterSlatSize is set to 0, then corresponding MasterKeyOffset or MasterSaltOffset must be ignored.

```
typedef struct DSCPValueInformation{
DWORD DSCPValueForAudioCalls;
}

typedef struct  PartitionInformation
{
DWORD CallerIDPartitionOffset;
DWORD CallerIDPartitionSize;
DWORD CalledIDPartitionOffset;
DWORD CalledIDPartitionSize;
```

```
DWORD ConnecetedIDPartitionOffset;
DWORD ConnecetedIDPartitionSize;
DWORD RedirectionIDPartitionOffset;
DWORD RedirectionIDPartitionSize;
DWORD RedirectingIDPartitionOffset;
DWORD RedirectingIDPartitionSize;
} PartitionInformation;


Struct ExtendedCallInfo
{
DWORD ExtendedCallReason ;
DWORD CallerIDURLOffset;// CallPartySipURLInfo
DWORD CallerIDURISize;
DWORD CalledIDURLOffset;// CallPartySipURLInfo
DWORD CalledIDURISize;
DWORD ConnectedIDURIOffset;// CallPartySipURLInfo
DWORD ConnectedIDURISize;
DWORD RedirectionIDURIOffset;// CallPartySipURLInfo
DWORD RedirectionIDURISize;
DWORD RedirectingIDURIOffset;// CallPartySipURLInfo
DWORD RedirectingIDURISize;
}


Struct CallPartySipURLInfo
{
DWORD dwUserOffset;  //sip user string
DWORDdwUserSize;
DWORDdwHostOffset; //host name string
DWORDdwHostSize;
DWORDdwPort;// integer port number
DWORD dwTransportType; // SIP_TRANS_TYPE
DWORD dwURLType;// SIP_URL_TYPE
}

enum {
    CTI_SIP_TRANSPORT_TCP=1,
    CTI_SIP_TRANSPORT_UDP,
    CTI_SIP_TRANSPORT_TLS
} SIP_TRANS_TYPE;
enum {
    CTI_NO_URL = 0,
    CTI_SIP_URL,
    CTI_TEL_URL
} SIP_URL_TYPE;

typedef struct CallAttributeInfo
{
DWORD CallAttributeType;
DWORD PartyDNOffset;
DWORD PartyDNSize;
DWORD PartyPartitionOffset;
DWORD PartyPartitionSize;
DWORD DeviceNameOffset;
DWORD DeviceNameSize;
DWORD OverallCallSecurityStatus;
}

typedef struct CallAttributeInfo_ExtA0
{
DWORD CallAttributeType;
DWORD PartyDNOffset;
DWORD PartyDNSize;
```

```
DWORD PartyPartitionOffset;
DWORD PartyPartitionSize;
DWORD DeviceNameOffset;
DWORD DeviceNameSize;
DWORD OverallCallSecurityStatus;
DWORD TransactionID;//Secure R & M
} CallAttributeInfo_ExtA0;

typedef struct CallAttributeInfo_ExtB0
{
CallAttributeInfo_ExtA0 attr_a0;
DWORD ActiveToneDirection;
} CallAttributeInfo_ExtB0;

typedef struct CCMCallHandleInformation
{
DWORD CCMCallID;
}

enum
{
    CallAttribute_Regular                    = 0,
    CallAttribute_SilentMonitorCall,
    CallAttribute_SilentMonitorCall_Target,
    CallAttribute_RecordedCall,
    CallAttribute_WhisperCoachingCall,
    CallAttribute_WhisperCoachingCall_Target,
    CallAttribute_Recorded_Automatic,
    CallAttribute_Recorded_AppInitiatedSilent,
    CallAttribute_Recorded_UserInitiatedFromApp,
    CallAttribute_Recorded_UserInitiatedFromDevice
} CallAttributeType
typedef struct CallSecurityStausInfo
{
DWORD CallSecurityStaus
} CallSecurityStausInfo
enum OverallCallSecurityStatus
{
OverallCallSecurityStatus_Unknown = 0,
OverallCallSecurityStatus_NotAuthenticated,
OverallCallSecurityStatus_Authenticated,
OverallCallSecurityStatus_Encrypted
};

typedef struct CPNInfo
{
DWORD CallerPartyNumberType;//refer to CiscoNumberType
DWORD CalledPartyNumberType;
DWORD ConnectedIdNumberType;
DWORD RedirectingPartyNumberType;
DWORD RedirectionPartyNumberType;
DWORD ModifiedCallingPartySize;
DWORD ModifiedCallingPartyOffset;
DWORD ModifiedCalledPartySize;
DWORD ModifiedCalledPartyOffset;
DWORD ModifiedConnectedIdSize;
DWORD ModifiedConnectedIdOffset;
DWORD ModifiedRedirectingPartySize;
DWORD ModifiedRedirectingPartyOffset;
DWORD ModifiedRedirectionPartySize;
DWORD ModifiedRedirectionPartyOffset;
DWORD GlobalizedCallingPartySize;
DWORD GlobalizedCallingPartyOffset;
} CPNInfo;
```

```
enum CiscoNumberType {
    NumberType_Unknown  = 0,          // UNKNOWN_NUMBER
    NumberType_International  = 1,    // INTERNATIONAL_NUMBER
    NumberType_National  = 2,         // NATIONAL_NUMBER
    NumberType_NetSpecificNum  = 3,   // NET_SPECIFIC_NUMBER
    NumberType_Subscriber  = 4,       // SUBSCRIBER_NUMBER
    NumberType_Abbreviated  = 6       // ABBREVIATED_NUMBER
};

typedef struct Cisco_LineCallInfo_Ext000A0000
{
     …
   //HuntList
    DWORD HuntPilotInfoOffset;//point to HuntPoiltInfo
    DWORD HuntPilotInfoSize;
    DWORD HuntPilotInfoCount;
    DWORD HuntPilotInfoElementFixedSize;
}CISCO_LINECALLINFO_EXT000A0000;

//HuntList
typedef struct HuntPilotInfo
{
DWORD CallingPartyHuntPilotDNOffset;
DWORD CallingPartyHuntPilotDNSize;
DWORD CallingPartyHuntPilotPartitionOffset;
DWORD CallingPartyHuntPilotPartitionSize;
DWORD CalledPartyHuntPilotDNOffset;
DWORD CalledPartyHuntPilotDNSize;
DWORD CalledPartyHuntPilotPartitionOffset;
DWORD CalledPartyHuntPilotPartitionSize;
DWORD ConnectedPartyHuntPilotDNOffset;
DWORD ConnectedPartyHuntPilotDNSize;
DWORD ConnectedPartyHuntPilotPartitionOffset;
DWORD ConnectedPartyHuntPilotPartitionSize;
}HuntPilotInfo;

typedef struct UniqueCallRefIDInfo
{
DWORD UniqueCallRefIDOffset;
DWORD UniqueCallRefIDSize;
} UniqueCallRefIDInfo;

typedef enum
{
TSPCallAttribute_Normal =                   0x00000000,
TSPCallAttribute_IntercomOriginator =       0x00000001,
TSPCallAttribute_IntercomTarget =           0x00000002,
TSPCallAttribute_SilentMonitorCall =        0x00000004,
TSPCallAttribute_SilentMonitorCall_Target = 0x00000008,
TSPCallAttribute_RecordedCall =             0x00000010,
TSPCallAttribute_ChaperoneCall =            0x00000020,
TSPCallAttribute_CallForwardAllSet =        0x00000040,
TSPCallAttribute_CallForwardAllClear =      0x00000080,
TSPCallAttribute_WhisperMonitorCall =       0x00000100,
TSPCallAttribute_WhisperMonitorCall_Target= 0x00000200,
TSPCallAttribute_BIBCall =              0x00000400,
TSPCallAttribute_ServerCall =0x00000800,
TSPCallAttribute_SendMediaToBIB =        0x00001000
} CallAttributeBits
```

Cisco TSP exposes the multimedia capability information of a linedevice in the devspecific data of LINEGETCALLINFO when LineGetCallInfo is invoked with Extension version 0x000D0000 or higher.

The calling party and the called party multimedia capability is exposed as a structure DeviceVideoCapInfo in the DevSpecific part. This structure contains three fields:

- deviceVideoCapability,
- telepresenceInfo, and,
- screenCount.

```
typedef struct DeviceVideoCapInfo{
    DWORD VideoCapStatus;
    DWORD TelepresenceInfo;
    DWORD ScreenCount;
} DeviceVideoCapInfo;
```

Data fields	Value
DeviceVideoCapability	DeviceVideoCapability field contains the type value defined in the following enumeration [CiscoDeviceVideoCapabilityInfo].
TelepresenceInfo	This field indicates if Telepresence is enabled on the device, defined in the following enumeration [CiscoDeviceTelepresenceInfo].
ScreenCount	This field indicated the number of screens present on the device.

```
enum CiscoDeviceVideoCapabilityInfo
{
    CiscoDeviceVideoCapability_None =          0x00000000,
    CiscoDeviceVideoCapability_Enabled =       0x00000001,
};

enum CiscoDeviceTelepresenceInfo
{
    CiscoDeviceTelepresence_None =          0x00000000,
    CiscoDeviceTelepresence_Enabled =       0x00000001,
};
```

Parameters

Parameter	Value
TapiCallerPartyUnicodeNameOffsetTapiCallerPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Caller party identifier name information, and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiCallerPartyLocale	The Unicode Caller party identifier name Locale information

Parameter	Value
TapiCallerPartyUnicodeNameOffsetTapiCallerPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Caller party identifier name information, and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiCallerPartyLocale	The Unicode Caller party identifier name Locale information
TapiCalledPartyUnicodeNameOffsetTapiCalledPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Called party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiCalledPartyLocale	The Unicode Called party identifier name locale information
TapiConnectedPartyUnicodeNameOffsetTapiConnectedPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Connected party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiConnectedPartyLocale	The Unicode Connected party identifier name locale information
TapiRedirectionPartyUnicodeNameOffsetTapiRedirectionPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Redirection party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiRedirectionPartyLocale	The Unicode Redirection party identifier name locale information
TapiRedirectingPartyUnicodeNameOffsetTapiRedirectingPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Redirecting party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiRedirectingPartyLocale	The Unicode Redirecting party identifier name locale information
SRTPKeyInfoStructureOffset	Point to SRTPKeyInfoStructure
SRTPKeyInfoStructureSize	Total size of SRTP information
SRTPKeyInfoStructureElementCount	Number of SRTPKeyInfoStructure element
SRTPKeyInfoStructureElementFixedSize	Fixed size of SRTPKeyInfoStructure
SecureMediaIndicator	Indicates whether media is secure and whether application is authorized for key information
MasterKeyOffsetMasterKeySize	The offset and size of SRTP MasterKey information

Parameter	Value
MasterSaltOffsetMasterSaltSize	The offset and size of SRTP MasterSaltKey information
AlgorithmID	Specifies negotiated SRTP algorithm ID
IsMKIPresent	Indicates whether MKI is present
KeyDerivationRate	Provides the KeyDerivationRate
DSCPValueForAudioCalls	The DSCP value for Audio Calls
CallerIDPartitionOffsetCallerIDPartitionSize	The size, in bytes, of the variably sized field that contains the Caller party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
CalledIDPartitionOffsetCalledIDPartitionSize	The size, in bytes, of the variably sized field that contains the Called party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
ConnectedIDPartitionOffsetConnecetedIDPartitionSize	The size, in bytes, of the variably sized field that contains the Connected party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectionIDPartitionOffsetRedirectionIDPartitionSize	The size, in bytes, of the variably sized field that contains the Redirection party identifier partition information, and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectingIDPartitionOffsetRedirectingIDPartitionSize	The size, in bytes, of the variably sized field that contains the Redirecting party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure

Parameter	Value
ExtendedCallReason	Presents all the last feature-related CTI Call reason code to the application as an extension to the standard reason codes that TAPI supports. This provides the feature-specific information per call. As phones that are running SIP are supported through CTI, new features can get introduced for phones that are running on SIP during releases. **Note** Be aware that this field is not backward compatible and can change as changes or additions are made in the SIP phone support for a feature. Applications should implement some default behavior to handle any unknown reason codes that might be provided through this field. For Refer, the reason code specified is CtiCallReason_Refer. For Replaces, the reason code specified is CtiCallReason_Replaces.
CallerIDURLOffsetCallerIDURLSize	The size, in bytes, of the variably sized field that contains the Caller party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
CalledIDURLOffsetCalledIDURLSize	The size, in bytes, of the variably sized field that contains the Called party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
ConnectedIDURLOffsetConnecetedIDURLSize	The size, in bytes, of the variably sized field that contains the Connected party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectionIDURLOffsetRedirectionIDURLSize	The size, in bytes, of the variably sized field that contains the Redirection party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectingIDURLOffsetRedirectingIDURLSize	The size, in bytes, of the variably sized field that contains the Redirecting party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
CallAttributeType	Identifies whether the following information (DN.Partition.DeviceName) is for a regular call, a monitoring call, a monitored call, or a recording call

Parameter	Value
PartyDNOffset, PartyDNSize,	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party DN information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
PartyPartitionOffset PartyPartitionSize	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party partition information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
DeviceNameOffset DeviceNameSize	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party device name and the offset, in bytes, from the beginning of the LINECALLINFO data structure
OverallCallSecurityStatus	The security status of the call for two-party calls and conference calls
CCMCallID	The Cisco Unified Communications Manager caller ID for each call leg
CallingPartyHuntPilotDNOffset CallingPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CallingPartyHuntPilotPartitionOffset CallingPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CalledPartyHuntPilotDNOffset CalledPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CalledPartyHuntPilotPartitionOffset CalledPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
ConnectedPartyHuntPilotDNOffset ConnectedPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
ConnectedPartyHuntPilotPartitionOffset ConnectedPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.

Parameter	Value
IsChaperoneCall	This field specifies whether the call is a chaperone call or not.

To indicate that partition information exists in the LINECALLINFO structure, the system fires a LINECALLINFOSTATE_DEVSPECIFIC event. The bit map indicating the change is defined as the following:

SLDST_SRTP_INFO	0x00000001
SLDST_QOS_INFO	0x00000002
SLDST_PARTITION_INFO	0x00000004
SLDST_EXTENDED_CALL_INFO	0x00000008
SLDST_CALL_ATTRIBUTE_INFO	0x00000010 //M&R
SLDST_CCM_CALL_ID	0x00000020 //M&R
SLDST_SECURITY_STATUS_INFO	0x00000040 //SecureConf
SLDST_NUMBER_TYPE_CHANGED	0x00000080 //CPN
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED	0x00000100 //CPN
SLDST_FAR_END_IP_ADDRESS_CHANGED	0x00000200//IPv6 new
SLDST_UNIQUE_CALL_REF_ID_INFO	0x00000400
SLDST_DEVICE_VIDEO_CAP_INFO	0x00000800
SLDST_MULTIMEDIA_STREAMS_INFO	0x00001000

```
LINEDEVSPECIFIC{
  hDevice = hcall //call handle for which the info has changed.
  dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA //indicates DevSpecific portion's
 changed
  dwParam2 = SLDST_SRTP_INFO | SLDST_QOS_INFO |SLDST_PARTITION_INFO |
SLDST_EXTENDED_CALL_INFO | SLDST_CALL_ATTRIBUTE_INFO|SLDST_CCM_CALLID|
SLDST_CALL_SECURITY_STATUS | SLDST_NUMBER_TYPE_CHANGED |
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED | SLDST_FAR_END_IP_ADDRESS_CHANGED |
SLDST_UNIQUE_CALL_REF_ID_INFO | SLDST_DEVICE_VIDEO_CAP_INFO |
SLDST_MULTIMEDIA_STREAMS_INFO
  dwParam3 = …
  dwParam3 will be security indicator if dwParam2 has bit set for SLDST_SRTP_INFO
}
```

LINECALLPARAMS

Cisco TSP implements several line device-specific extensions that require applications to use LINECALLPARAMS structure to pass relevant data in the lineMakeCall request.

Details

With extention 0x00080001 feature priority is introduced for DoNotDisturb-Reject feature. Feature priority can be specified in DevSpecific part of LINECALLPARAMS (dwDevSpecificSize and dwDevSpecificOffset) as

```
typedef struct LineParamas
{
     DWORD FeaturePriority;
} LINE_PRAMS;
```

Starting with extention 0x000D0000 Feature Priority data is included in Cisco_CallParamsDevSpecific structure that replaces LineParams structure. The Cisco_CallParamsDevSpecific structure is defined in CiscoLineDevSpecificMsg.h header file as follows:

```
typedef struct Cisco_CallParamsDevSpecific_tag
{
     DWORD CallPriority;
     DWORD DevSpecificFlags;
} Cisco_CallParamsDevSpecific;
```

The DevSpecificFlags field in that structure is used to identify a specific feature and can be set to one of the following:

```
#define Cisco_CALLPARAMS_DEVSPECIFICFLAGS_PRIORITYCALL      0x00000001
#define Cisco_CALLPARAMS_DEVSPECIFICFLAGS_PERSISTENTCALL    0x00000002
#define Cisco_CALLPARAMS_DEVSPECIFICFLAGS_ANNOUNCEMENTCALL  0x00000004
```

Cisco_CALLPARAMS_DEVSPECIFICFLAGS_PRIORITYCALL indicates the feature priority call for DoNotDisturb-Reject feature. The CallPriority field in Cisco_CallParamsDevSpecific structure should be used to specify feature priority.

Cisco_CALLPARAMS_DEVSPECIFICFLAGS_PERSISTENTCALL indicates that Persistent Call is to be created. In this case, two other LINECALLPARAMS fields are used:

- CallingPartyID (dwCallingPartyIDSize and dwCallingPartyIDOffset) is used to specify calling-party ID
- DisplayabeAddress (dwDisplayableAddress Size and dwDisplayableAddress Offset) is used to specify calling-party ID name

Cisco_CALLPARAMS_DEVSPECIFICFLAGS_ANNOUNCEMENTCALL indicates that Announcement call is to be created. In that case CallData field in the LINECALLPARAMS structure is used to specify an announcement ID. Announcement ID (or media-contmt ID) is a string with a maximum length defined in CiscoLineDevSpecificMsg.h as MAX_CISCO_TSP_MEDIA_CONTENT_ID_SIZE.

LINEDEVSTATUS

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINEDEVSTATUS data structure for those extensions. Cisco TSP defines the DevSpecific area layout in the Cisco_LineDevStatus_Ext structure in the CiscoLineDevSpecificMsg.h header file. The extension version in which the data was introduced provides basis for how the data in that structure is organized.

```
//   LINEDEVSTATUS Dev Specific extention   //
typedef struct Cisco_LineDevStatus_Ext
{
    Cisco_LineDevStatus_Ext00060000  ext60;
    Cisco_LineDevStatus_Ext00070000  ext70;
    Cisco_LineDevStatus_Ext00080000  ext80;
} CISCO_LINEDEVSTATUS_EXT;

typedef struct Cisco_LineDevStatus_Ext00080000
{
CISCOLINEDEVSTATUS_DONOTDISTURB doNotDisturbStatus;
} CISCO_LINEDEVSTATUS_EXT00080000;

typedef struct CiscoLineDevStatus_DoNotDisturb
{
DWORD m_LineDevStatus_DoNotDisturbOption;
DWORD m_LineDevStatus_DoNotDisturbStatus;
} CISCOLINEDEVSTATUS_DONOTDISTURB;
```

For a specific line device, the extension area will include a portion of this structure, starting from the beginning and up to the extension version that an application negotiated.

Detail

The individual extension version substructure definitions follow:

```
//    LINEDEVSTATUS 00060000  extention    //
typedef struct Cisco_LineDevStatus_Ext00060000
{
    DWORD dwSupportedEncoding;
} CISCO_LINEDEVSTATUS_EXT00060000;

//    LINEDEVSTATUS 00070000  extention    //
typedef struct Cisco_LineDevStatus_Ext00070000
{
    char lpszAlternateScript[MAX_ALTERNATE_SCRIPT_SIZE];
    // An empty string means there  is no alternate script configured
    // or the phone does not support alternate scripts
} CISCO_LINEDEVSTATUS_EXT00070000;

//    LINEDEVSTATUS 00080000  extention    //
//   Status extention 00080000 Data Structure//
typedef struct Cisco_LineDevStatus_Ext00080000
{
CISCOLINEDEVSTATUS_DONOTDISTURB doNotDisturbStatus;
} CISCO_LINEDEVSTATUS_EXT00080000;

typedef struct CiscoLineDevStatus_DoNotDisturb
{
    DWORD m_LineDevStatus_DoNotDisturbOption;
    DWORD m_LineDevStatus_DoNotDisturbStatus;
} CISCOLINEDEVSTATUS_DONOTDISTURB;
```

You can find additional information on the DevSpecific structure layout and data in the CiscoLineDevSpecificMsg.h header file.

The CiscoLineDevStatus_DoNotDisturb structure belongs to the LINEDEVSTATUS_DEV_SPECIFIC_DATA structure and gets used to reflect the current state of the Do Not Disturb feature.

Parameters

DWORD dwSupportEncoding

This parameter indicates the Support Encoding for the Unicode Party names that are being sent in device-specific extension of the LINECALLINFO structure.

The typical values could be

```
enum {
UnknownEncoding = 0,// Unknown encoding
NotApplicableEncoding = 1,// Encoding not applicable to this device
AsciiEncoding = 2,          // ASCII encoding
Ucs2UnicodeEncoding = 3     // UCS-2 Unicode encoding
}
```

Note Be aware that the dwSupportedEncoding extension is only available if extension version 0x00060000 or higher is negotiated.

LPCSTR lpszAlternateScript

This parameter specifies the alternate script that the device supports. An empty string indicates the device does not support or is not configured with an alternate script.

The only supported script in this release is "Kanji" for the Japanese locale.

m_LineDevStatus_DoNotDisturbOption

This field contains DND option that is configured for the device and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOption {    DoNotDisturbOption_NONE      = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT    = 2
};

m_LineDevStatus_DoNotDisturbStatus field contains current DND status on the
device and can be one of the following enum values:

enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN  = 0,
    DoNotDisturbStatus_ENABLED  = 1,
    DoNotDisturbStatus_DISABLED = 2
};
```

Note Be aware that this extension is only available if extension version 8.0 (0x00080000) or higher is negotiated.

CCiscoLineDevSpecific

This section provides information on how to perform Cisco Unified TAPI specific functions with the CCiscoLineDevSpecific class, which represents the parent class to all the following classes. It comprises a virtual class and is provided here for informational purposes.

```
CCiscoLineDevSpecific|
+--CCiscoLineDevSpecificMsgWaiting
|
+--CCiscoLineDevSpecificMsgWaitingDirn
|
+--CCiscoLineDevSpecificUserControlRTPStream
|
+--CCiscoLineDevSpecificSetStatusMsgs
|
+--CCiscoLineDevSpecificSwapHoldSetupTransfer
|
+--CCiscoLineDevSpecificRedirectResetOrigCalled
|
+--CCiscoLineDevSpecificRedirectSetOrigCalled
|
+--CCiscoLineDevSpecificPortRegistrationPerCall
|
+--CCiscoLineDevSpecificSetRTPParamsForCall
|
+--CCiscoLineDevSpecificJoin
|
+--CCiscoLineDevSpecificRedirectFACCMC
|
+--CCiscoLineDevSpecificBlindTransferFACCMC
|
+--CCiscoLineDevSpecificCTIPortThirdPartyMonitor
|
+--CCiscoLineDevSpecificUserSetSRTPAlgorithmID
|
+--CCiscoLineDevSpecificSendLineOpen
|
+--CCiscoLineDevSpecificAcquire
|
+--CCiscoLineDevSpecificDeacquire
|
+--CCiscoLineDevSpecificStartCallMonitoring
|
+--CCiscoLineDevSpecificStartCallRecording
|
+--CCiscoLineDevSpecificStopCallRecording
|
+--CCiscoLineSetIntercomSpeeddial
|
+--CCiscoLineIntercomTalkback
|
+--CciscoSetupTransferWithoutMedia
|
+--CCiscoLineDevSpecificSetMsgSummary
|
+--CCiscoLineDevSpecificSetMsgSummaryDirn
|
+--CCiscoLineDevSpecificSetRTPParamsForCallIPv6
|
+--CCiscoLineDevSpecificSetIPv6AddressAndMode
|
+--CCiscoLineDevSpecificDirectTransfer
|
+--CCiscoLineDevSpecificRegisterCallPickupGroupForNotification
|
+--CCiscoLineDevSpecificUnRegisterCallPickupGroupForNotification
|
+--CCiscoLineDevSpecificCallPickupRequest
|
+--CCiscoLineDevSpecificPlaytone
```

```
|
+--CCiscoLineDevSpecificStartSendMediaToBIBRequest
|
+--CCiscoLineDevSpecificStopSendMediaToBIBRequest
|
+--CCiscoLineDevSpecificMonitoringUpdateMode
|
+--CCiscoLineDevSpecificEnableFeatureSupport
|
+--CCiscoLineRedirectWithFeaturePriority
|
+--CciscoLineDevSpecificAddRemoteDestination
|
+--CciscoLineDevSpecificUpdateRemoteDestination
|
+--CciscoLineDevSpecificRemoveRemoteDestination
|
+--CciscoLineDevSpecificHoldEx
|
```

Header File

The file CiscoLineDevSpecific.h contains the constant, structure, and class definition for the Cisco line device-specific classes.

Class Detail

```
class CCiscoLineDevSpecific
  {
  public:
    CCicsoLineDevSpecific(DWORD msgType);
    virtual ~CCiscoLineDevSpecific();
    DWORD GetMsgType(void) const {return m_MsgType;}
    void* lpParams() {return &m_MsgType;}
    virtual DWORD dwSize() = 0;
  private:
    DWORD m_MsgType;
  };
```

Functions

lpParms()

You can use function to obtain the pointer to the parameter block.

dwSize()

Function will give the size of the parameter block area.

Parameter

m_MsgType

Specifies the type of message.

Subclasses

Each subclass of CCiscoLineDevSpecific includes a different value that is assigned to the parameter m_MsgType. If you are using C instead of C++, this represents the first parameter in the structure.

Enumeration

The CiscoLineDevSpecificType enumeration provides valid message identifiers.

```
enum CiscoLineDevSpecificType
{
SLDST_MSG_WAITING = 1,
SLDST_MSG_WAITING_DIRN,
SLDST_USER_CRTL_OF_RTP_STREAM,
SLDST_SET_STATUS_MESSAGES,
SLDST_NUM_TYPE,
SLDST_SWAP_HOLD_SETUP_TRANSFER, // Not Supported in CiscoTSP 3.4 and Beyond
SLDST_REDIRECT_RESET_ORIG_CALLED,
SLDST_REDIRECT_SET_ORIG_CALLED,
SLDST_USER_RECEIVE_RTP_INFO,
SLDST_USER_SET_RTP_INFO,
SLDST_JOIN,
SLDST_REDIRECT_FAC_CMC,
SLDST_BLIND_TRANSFER_FAC_CMC,
SLDST_CTI_PORT_THIRD_PARTY_MONITOR,
SLDST_ACQUIRE,
SLDST_DE_ACQUIRE,
SLDST_USER_SET_SRTP_ALGORITHM_ID,
SLDST_SEND_LINE_OPEN,
SLDST_START_CALL_MONITORING,
SLDST_START_CALL_RECORDING,
SLDST_STOP_CALL_RECORDING,
SLDST_LINE_SET_INTERCOM_SPEEDDIAL,
SLDST_LINE_INTERCOM_TALKBACK,
SLDST_REDIRECT_WITH_FEATURE_PRIORITY,
SLDST_USER_SET_RTP_INFO_IPv6,
SLDST_USER_SET_IPv6_ADDRESS_AND_MODE,
SLDST_SETUP_TRANSFER_WITHOUT_MEDIA,
SLDST_DIRECT_TRANSFER,
SLDST_MSG_SUMMARY,
SLDST_MSG_SUMMARY_DIRN,
SLDST_CALLPICKUP_GROUP_REGISTER,
SLDST_CALLPICKUP_GROUP_UNREGISTER,
SLDST_CALLPICKUP_CALL,
SLDST_PLAY_TONE,
SLDST_START_SEND_MEDIA_TO_BIB,
SLDST_STOP_SEND_MEDIA_TO_BIB,
SLDST_UPDATE_MONITOR_MODE,
SLDST_ENABLE_FEATURE_SUPPORT,
SLDST_ADD_REMOTE_DESTINATION,
SLDST_REMOVE_REMOTE_DESTINATION,
SLDST_UPDATE_REMOTE_DESTINATION,
SLDST_HOLD_EX

};
```

Message Waiting

The CCiscoLineDevSpecificMsgWaiting class turns the message waiting lamp on or off for the line that the hLine parameter specifies.

Note This extension does not require an extension version to be negotiated.

```
CCiscoLineDevSpecific
  |
  +--CCiscoLineDevSpecificMsgWaiting
```

Class Detail

```
class CCiscoLineDevSpecificMsgWaiting : public CCiscoLineDevSpecific
{
 public:
  CCiscoLineDevSpecificMsgWaiting() : CCiscoLineDevSpecific(SLDST_MSG_WAITING){}
  virtual ~CCiscoLineDevSpecificMsgWaiting() {}
  virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
  DWORD m_BlinkRate;
};
```

Parameters

DWORD m_MsgType

Equals SLDST_MSG_WAITING.

DWORD m_BlinkRate

Any supported PHONELAMPMODE_constants that are specified in the phoneSetLamp() function.

Note Cisco Unified IP Phone 7900 Series supports only PHONELAMPMODE_OFF and PHONELAMPMODE_STEADY

Message Waiting Dirn

The CCiscoLineDevSpecificMsgWaitingDirn class turns the message waiting lamp on or off for the line that a parameter specifies and remains independent of the hLine parameter.

Note This extension does not require an extension version to be negotiated.

```
CCiscoLineDevSpecific
  |
  +--CCiscoLineDevSpecificMsgWaitingDirn
```

Class Detail

```
class CCiscoLineDevSpecificMsgWaitingDirn : public CCiscoLineDevSpecific
{
 public:
  CCiscoLineDevSpecificMsgWaitingDirn() :
    CCiscoLineDevSpecific(SLDST_MSG_WAITING_DIRN) {}
  virtual ~CCiscoLineDevSpecificMsgWaitingDirn() {}
  virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
  DWORD m_BlinkRate;
  char m_Dirn[25];
};
```

Parameters

DWORD m_MsgType

Specifies SLDST_MSG_WAITING_DIRN.

DWORD m_BlinkRate

As in the CCiscoLineDevSpecificMsgWaiting message.

Note Cisco Unified IP Phone 7900 Series supports only PHONELAMPMODE_OFF and PHONELAMPMODE_STEADY

char m_Dirn[25]

The directory number for which the message waiting lamp should be set.

Message Summary

Use the CCiscoLineDevSpecificSetMsgSummary class to turn the message waiting lamp on or off as well as to provide voice and fax message counts for the line specified by the hLine parameter.

Note Be aware that this extension does not require an extension version to be negotiated.

```
CCiscoLineDevSpecific
  |
  +--CCiscoLineDevSpecificSetMsgSummary
```

Class Detail

```
class CCiscoLineDevSpecificSetMsgSummary : public CCiscoLineDevSpecific
{
public:
  CCiscoLineDevSpecificSetMsgSummary() : CCiscoLineDevSpecific(SLDST_MSG_SUMMARY){}

  virtual ~CCiscoLineDevSpecificSetMsgSummary() {}
  virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
```

```
  DWORD m_BlinkRate;
  MSG_SUMMARY m_MessageSummary;
};
```

Parameters

DWORD m_MsgType

equals SLDST_MSG_SUMMARY.

DWORD m_BlinkRate

is any supported PHONELAMPMODE_constants specified in the phoneSetLamp() function.

MSG_SUMMARY m_MessageSummary

A data structure with the following format:

```
typedef struct {
DWORD m_voiceCounts;  // indicates if new voice counts are
                      //  provided. True = counts will be displayed
                      //  on supported phones.
DWORD m_totalNewVoiceMsgs; // specifies the total number of new
                      // voice messages. This number includes all
                      // the high and normal priority voice
                      // messages that are new.
DWORD m_totalOldVoiceMsgs; // specifies the total number of old
                      // voice messages. This number includes all
                      // high and normal priority voice messages
                      // that are old.
DWORD m_highPriorityVoiceCounts; // indicates if old voice
                      // counts are provided. True = counts will be
                      // displayed on supported phones.
DWORD m_newHighPriorityVoiceMsgs; //specifies the number of new
                      // high priority voice messages.
DWORD m_oldHighPriorityVoiceMsgs; //specifies the number of old
                      // high priority voice messages.
DWORD m_faxCounts;    // indicates if new fax counts are
                      // provided. True = counts will be displayed
                      // on supported phones.
DWORD m_totalNewFaxMsgs; // specifies the total number of new
                      // fax messages. This number includes all
                      // the high and normal priority fax
                      // messages that are new.
DWORD m_totalOldFaxMsgs; // specifies the total number of old
                      // fax messages. This number includes all
                      // high and normal priority fax messages
                      // that are old.
DWORD m_highPriorityFaxCounts; // indicates if old fax counts
                      //  are provided. True = counts will be
                      //  displayed on supported phones.
DWORD m_newHighPriorityFaxMsgs; // specifies the number of new
                      // high priority fax messages.
DWORD m_oldHighPriorityFaxMsgs; // specifies the number of old
                      // high priority fax messages.
       } MSG_SUMMARY;
```

Message Summary Dirn

Use the CCiscoLineDevSpecificSetMsgSummaryDirn class to turn the message waiting lamp on or off and to provide voice and fax message counts for the line specified by a parameter and is independent of the hLine parameter.

Note Be aware that this extension does not require an extension version to be negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificSetMsgSummaryDirn
```

Class Detail

```
class CCiscoLineDevSpecificSetMsgSummaryDirn : public CCiscoLineDevSpecific
{
public:
  CCiscoLineDevSpecificSetMsgSummaryDirn() :
CCiscoLineDevSpecific(SLDST_MSG_SUMMARY_DIRN) {}
  virtual ~CCiscoLineDevSpecificSetMsgSummaryDirn() {}
  virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
  DWORD m_BlinkRate;
  char m_Dirn[25];
  MSG_SUMMARY m_MessageSummary;
};
```

Parameters

DWORD m_MsgType

equals SLDST_MSG_SUMMARY_DIRN.

DWORD m_BlinkRate

is as in the CCiscoLineDevSpecificSetMsgSummary message.

char m_Dirn[25]

is the directory number for which the message waiting lamp should be set.

MSG_SUMMARY m_MessageSummary

A data structure with the following format:

```
  typedef struct {
 DWORD m_voiceCounts; // indicates if new voice counts are
                      //  provided. True = counts will be displayed
                      //  on supported phones.
 DWORD m_totalNewVoiceMsgs; // specifies the total number of new
                      // voice messages. This number includes all
                      // the high and normal priority voice
                      // messages that are new.
 DWORD m_totalOldVoiceMsgs; // specifies the total number of old
                      // voice messages. This number includes all
                      // high and normal priority voice messages
```

```
                        // that are old.
 DWORD m_highPriorityVoiceCounts; // indicates if old voice
                        // counts are provided. True = counts will be
                        // displayed on supported phones.
 DWORD m_newHighPriorityVoiceMsgs; //specifies the number of new
                        // high priority voice messages.
 DWORD m_oldHighPriorityVoiceMsgs; //specifies the number of old
                        // high priority voice messages.
 DWORD m_faxCounts; // indicates if new fax counts are
                        //  provided. True = counts will be displayed
                        //  on supported phones.
 DWORD m_totalNewFaxMsgs; // specifies the total number of new
                        // fax messages. This number includes all
                        // the high and normal priority fax
                        // messages that are new.
 DWORD m_totalOldFaxMsgs; // specifies the total number of old
                        // fax messages. This number includes all
                        // high and normal priority fax messages
                        // that are old.
 DWORD m_highPriorityFaxCounts; // indicates if old fax counts
                        //  are provided. True = counts will be
                        //  displayed on supported phones.
 DWORD m_newHighPriorityFaxMsgs; // specifies the number of new
                        // high priority fax messages.
 DWORD m_oldHighPriorityFaxMsgs; // specifies the number of old
                        // high priority fax messages.
       } MSG_SUMMARY;
```

Audio Stream Control

The CCiscoLineDevSpecificUserControlRTPStream class controls the audio stream of a line. To use this class you must call the lineNegotiateExtVersion API before opening the line. When lineNegotiateExtVersion is called ensure the highest bit is set on both the dwExtLowVersion and dwExtHighVersion parameters. This causes the call to lineOpen to behave differently. The line does not actually open, but waits for a lineDevSpecific call to complete the open with more information. The CCiscoLineDevSpecificUserControlRTPStream class provides the extra information that is required.

```
CCiscoLineDevSpecific
  |
  +--CCiscoLineDevSpecificUserControlRTPStream
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened (OR 0x80000000 with the dwExtLowVersion and dwExtHighVersion parameters).
2. Call lineOpen for the deviceID of the line that is to be opened.
3. Call lineDevSpecific with a CCiscoLineDevSpecificUserControlRTPStream message in the lpParams parameter.

Class Detail

```
class CCiscoLineDevSpecificUserControlRTPStream : public CCiscoLineDevSpecific
 {
  public:
```

```
 CCiscoLineDevSpecificUserControlRTPStream() :
   CCiscoLineDevSpecific(SLDST_USER_CRTL_OF_RTP_STREAM),
   m_ReceiveIP(-1),
   m_ReceivePort(-1),
   m_NumAffectedDevices(0)
    {
    memset(m_AffectedDeviceID, 0, sizeof(m_AffectedDeviceID));
    }
 virtual ~CCiscoLineDevSpecificUserControlRTPStream() {}
 DWORD m_ReceiveIP;   // UDP audio reception IP
 DWORD m_ReceivePort; // UDP audio reception port
 DWORD m_NumAffectedDevices;
 DWORD m_AffectedDeviceID[10];
 DWORD m_MediaCapCount;
 MEDIA_CAPS m_MediaCaps;
 virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
};
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_CRTL_OF_RTP_STREAM

DWORD m_ReceiveIP:

The RTP audio reception IP address in network byte order

DWORD m_ReceivePort:

The RTP audio reception port in network byte order

DWORD m_NumAffectedDevices:

The TSP returns this value. It contains the number of deviceIDs in the m_AffectedDeviceID array that are valid. Any device with multiple directory numbers that are assigned to it will have multiple TAPI lines, one per directory number.

DWORD m_AffectedDeviceID[10]:

The TSP returns this value. It contains the list of deviceIDs for any device that is affected by this call. Do not call lineDevSpecific for any other device in this list.

DWORD m_mediaCapCount

The number of codecs that are supported for this line.

MEDIA_CAPS m_MediaCaps -

A data structure with the following format:

```
typedef struct {
    DWORD MediaPayload;
    DWORD MaxFramesPerPacket;
    DWORD G723BitRate;
} MEDIA_CAPS[MAX_MEDIA_CAPS_PER_DEVICE];
```

This data structure defines each codec that is supported on a line. The limit specifies 18. The following description shows each member in the MEDIA_CAPS data structure:

MediaPayload specifies an enumerated integer that contains one of the following values:

```
enum    {
Media_Payload_G711Alaw64k = 2,
Media_Payload_G711Alaw56k = 3, // "restricted"
Media_Payload_G711Ulaw64k = 4,
Media_Payload_G711Ulaw56k = 5, // "restricted"
Media_Payload_G722_64k = 6,
Media_Payload_G722_56k = 7,
Media_Payload_G722_48k = 8,
Media_Payload_G7231 = 9,
Media_Payload_G728 = 10,
Media_Payload_G729 = 11,
Media_Payload_G729AnnexA = 12,
Media_Payload_G729AnnexB = 15,
Media_Payload_G729AnnexAwAnnexB = 16,
Media_Payload_GSM_Full_Rate = 18,
Media_Payload_GSM_Half_Rate = 19,
Media_Payload_GSM_Enhanced_Full_Rate = 20,
Media_Payload_Wide_Band_256k = 25,
Media_Payload_Data64 = 32,
Media_Payload_Data56 = 33,
Media_Payload_GSM = 80,
Media_Payload_G726_32K = 82,
Media_Payload_G726_24K = 83,
Media_Payload_G726_16K = 84,
// Media_Payload_G729_B = 85,
// Media_Payload_G729_B_LOW_COMPLEXITY = 86,
}   Media_PayloadType;
```

Read MaxFramesPerPacket as MaxPacketSize. It specifies a 16-bit integer that indicates the maximum desired RTP packet size in milliseconds. Typically, this value gets set to 20.

G723BitRate specifies a 6-byte field that contains either the G.723.1 information bit rate, or it gets ignored. The following list provides values for the G.723.1 field values:

```
enum    {
    Media_G723BRate_5_3 = 1, //5.3Kbps
    Media_G723BRate_6_4 = 2  //6.4Kbps
    }   Media_G723BitRate;
```

Set Status Messages

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificSetStatusMsgs
```

Description

Use the CCiscoLineDevSpecificSetStatusMsgs class to turn on or off the status messages for the line that the hLine parameter specifies. The Cisco Unified TSP supports the following flags:

- DEVSPECIFIC_MEDIA_STREAM—Setting this flag on a line turns on the reporting of media streaming messages for that line. Clearing this flag turns off the reporting of media streaming messages for that line.
- DEVSPECIFIC_CALL_TONE_CHANGED—Setting this flag on a line turns on the reporting of call tone changed events for that line. Clearing this flag turns off the reporting of call tone changed events for that line.

- DEVSPECIFIC_SILENT_MONITORING_TERMINATED—Setting this flag on a line turns on the reporting of Monitoring Session Terminated Event messages for that line. Clearing this flag turns off the reporting of Monitoring Session Terminated Event Messages for that line.
- DEVSPECIFIC_GET_IP_PORT—Setting this flag on a line turns on the reporting of Get IP and Port Notification Event messages for that line. Clearing this flag turns off the reporting of Get IP and Port Notification Event Messages for that line.
- DEVSPECIFIC_HOLD_REVERSION—Setting this flag on a line causes the application to receive a LINE_DEVSPECIFIC(dwParam1 = SLDSMT_HOLD_REVERSION) when a hold reversion happens on a held call. Clearing this flag on a line turns off the reporting of the LINE_DEVSPECIFIC(dwParam1 = SLDSMT_HOLD_REVERSION) event.
- DEVSPECIFIC_IDLE_TRANSFER_REASON—Setting this flag on a line causes the reason to be reported as LINECALLREASON_TRANSFER when calls go to the LINECALLSTATE_IDLE state after a transfer is completed at the transfer controller. Clearing this flag on a line causes the reason to be reported as LINECALLREASON_DIRECT when calls go to the LINECALLSTATE_IDLE state after a transfer is completed at the transfer controller.
- DEVSPECIFIC_SPEEDDIAL_CHANGED—Setting this flag on a line causes a LINE_DEVSPECIFIC(dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED, dwParam2 = LPCT_INTERCOM_LINE, and dwParam3 = CiscoIntercomLineChangeResult) to be fired to the application when there is a change in the database or the application overwrites the speed dial setting. Clearing this flag turns off the reporting of the LINE_DEVSPECIFIC(dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED, dwParam2 = LPCT_INTERCOM_LINE, and dwParam3 = CiscoIntercomLineChangeResult) event.
- DEVSPECIFIC_DONOTDISTURB_CHANGED—Setting this flag on a line causes a LINE_DEVSPECIFICFEATURE(dwParam1 = PHONEBUTTONFUNCTION_DONOTDISTURB, dwParam2 = typeOfChange, and dwParam3 = currentValue) to be fired to the application when there is a change in the DND configuration or status. Clearing this flag turns off the reporting of the LINE_DEVSPECIFICFEATURE(dwParam1 = PHONEBUTTONFUNCTION_DONOTDISTURB, dwParam2 = typeOfChange, and dwParam3 = currentValue) event.
- DEVSPECIFIC_DISPLAYABLE_ADDRESS—Setting this flag on a line causes the DisplayableAddress field in LINECALLINFO to be filled with the latest called partyDN/ASCCI name/Unicode name/Partition (separated by ":"). Clearing this flag causes the DisplayableAddress field in LINECALLINFO to be empty.
- DEVSPECIFIC_DEVICE_STATE—Setting this flag gets the accumulative state of all the lines on the device and with the state being fired to the application using the LINE_DEVSPECIFIC(dwParam1 = SLDSMT_DEVICE_STATE, dwParam2 = State) events. Clearing this flag turns off the reporting of the accumulative state of all the lines on the device.

  The DEVSPECIFIC_DEVICE_STATE state is defined as:

```
enum lineDeviceState{
    lineDeviceState_UNKNOWN =   0,
    lineDeviceState_ACTIVE =    1,
    lineDeviceState_ALERTING =  2,
    lineDeviceState_HELD =      3,
    lineDeviceState_WHISPER =   4,
    lineDeviceState_IDLE =      5
};
```

- DEVSPECIFIC_PARK_MONITORING—Setting this flag on a line causes the Park Monitoring events to be fired to the application. Clearing this flag turns off the reporting of the Park Monitoring events. For more information, see Park Monitoring, on page 78.
- DEVSPECIFIC_OTHER_DEVICE_STATE_NOTIFY—Setting this flag on a line notifies the application about the non-opened device state changes. Clearing this flag turns off the reporting of the other non-opened device state changes. For more information, see Other-Device State Notification, on page 77.

Note This extension only applies if extension version 0x00020001 or higher is negotiated.

Class Detail

```
class CCiscoLineDevSpecificSetStatusMsgs : public CCiscoLineDevSpecific
{
public:
CCiscoLineDevSpecificSetStatusMsgs() :
CCiscoLineDevSpecific(SLDST_SET_STATUS_MESSAGES) {}
virtual ~CCiscoLineDevSpecificSetStatusMsgs() {}
DWORD m_DevSpecificStatusMsgsFlag;
virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
};
```

Parameters

DWORD m_MsgType

Equals SLDST_SET_STATUS_MESSAGES.

DWORD m_DevSpecificStatusMsgsFlag

Identifies which status changes cause a LINE_DEVSPECIFIC message to be sent to the application.

The supported values follow:

```
#define DEVSPECIFIC_MEDIA_STREAM0x00000001#define
DEVSPECIFIC_CALL_TONE_CHANGED0x00000002
#define CALL_DEVSPECIFIC_RTP_EVENTS0x00000003
#define DEVSPECIFIC_IDLE_TRANSFER_REASON              0x00000004
#define DEVSPECIFIC_HOLD_REVERSION                    0x00000008
#define DEVSPECIFIC_SPEEDDIAL_CHANGED0x00000010
#define DEVSPECIFIC_DONOTDISTURB_CHANGED0x00000020
#define DEVSPECIFIC_DISPLAYABLE_ADDRESS0x00000040
#define DEVSPECIFIC_PARK_MONITORING0x00000080
#define DEVSPECIFIC_DEVICE_STATE0x00000100
#define DEVSPECIFIC_SILENT_MONITORING_TERMINATED0x00000200
#define DEVSPECIFIC_OTHER_DEVICE_STATE_NOTIFY0x00000400
#define DEVSPECIFIC_GET_IP_PORT0x00000800
```

Swap-Hold/SetupTransfer

Note Cisco Unified TSP 4.0 and later do not support this.

The CCiscoLineDevSpecificSwapHoldSetupTransfer class gets used to perform a SetupTransfer between a call that is in CONNECTED state and a call that is in the ONHOLD state. This function changes the state of the connected call to ONHOLDPENDTRANSFER state and the ONHOLD call to CONNECTED state. This allows a CompleteTransfer to be performed on the two calls. In Cisco Unified TSP 4.0 and later, the TSP allows applications to use lineCompleteTransfer() to transfer the calls without having to use the CCiscoLineDevSpecificSwapHoldSetupTransfer function. Therefore, this function returns LINEERR_OPERATIONUNAVAIL in Cisco Unified TSP 4.0 and beyond.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificSwapHoldSetupTransfer
```

Note This extension only applies if extension version 0x00020002 or higher is negotiated.

Class Details

```
class CCiscoLineDevSpecificSwapHoldSetupTransfer : public CCiscoLineDevSpecific
    {
    public:
      CCiscoLineDevSpecificSwapHoldSetupTransfer() :
CCiscoLineDevSpecific(SLDST_SWAP_HOLD_SETUP_TRANSFER) {}
      virtual ~CCiscoLineDevSpecificSwapHoldSetupTransfer() {}
      DWORD heldCallID;
      virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out
 the
virtual function table pointer
    };
```

Parameters

DWORD m_MsgType

Equals SLDST_SWAP_HOLD_SETUP_TRANSFER.

DWORD heldCallID

Equals the callid of the held call that is returned in dwCallID of LPLINECALLINFO.

HCALL hCall (in lineDevSpecific parameter list)

Equals the handle of the connected call.

Redirect Reset Original Called ID

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificRedirectResetOrigCalled
```

Description

The CCiscoLineDevSpecificRedirectResetOrigCalled class redirects a call to another party while it resets the original called ID of the call to the destination of the redirect.

Note This extension only applies if extension version 0x00020003 or higher is negotiated.

Class Details

```
class CCiscoLineDevSpecificRedirectResetOrigCalled: public CCiscoLineDevSpecific
    {
    public:
      CCiscoLineDevSpecificRedirectResetOrigCalled:
CCiscoLineDevSpecific(SLDST_REDIRECT_RESET_ORIG_CALLED) {}
      virtual ~CCiscoLineDevSpecificRedirectResetOrigCalled{}
      char m_DestDirn[25]; //redirect destination address
      virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out
 the
virtual function table pointer
    };
```

Parameters

DWORD m_MsgType

Equals SLDST_REDIRECT_RESET_ORIG_CALLED.

DWORD m_DestDirn

Equals the destination address where the call needs to be redirected.

HCALL hCall (In lineDevSpecific parameter list)

Equals the handle of the connected call.

Port Registration per Call

The CCiscoLineDevSpecificPortRegistrationPerCall class registers the CTI Port for the RTP parameters on a per-call basis. With this request, the application receives the new lineDevSpecific event that requests that it needs to set the RTP parameters for the call.

To use this class, ensure the lineNegotiateExtVersion API is called before opening the line. When calling lineNegotiateExtVersion, ensure the highest bit is set on both the dwExtLowVersion and dwExtHighVersion parameters.

This causes the call to lineOpen to behave differently. The line does not actually open, but waits for a lineDevSpecific call to complete the open with more information. The extra information required is provided in the CciscoLineDevSpecificPortRegistrationPerCall class.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificPortRegistrationPerCall
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened (or 0x80000000 with the dwExtLowVersion and dwExtHighVersion parameters)
2. Call lineOpen for the deviceID of the line that is to be opened.
3. Call lineDevSpecific with a CciscoLineDevSpecificPortRegistrationPerCall message in the lpParams parameter.

Note This extension is only available if the extension version 0x00040000 or higher gets negotiated.

Class Details

```
class CCiscoLineDevSpecificPortRegistrationPerCall: public CCiscoLineDevSpecific
{
public:
   CCiscoLineDevSpecificPortRegistrationPerCall () :
   CCiscoLineDevSpecific(SLDST_USER_RECEIVE_RTP_INFO),
   m_RecieveIP(-1), m_RecievePort(-1), m_NumAffectedDevices(0)
   {
   memset((char*)m_AffectedDeviceID, 0, sizeof(m_AffectedDeviceID));
   }

   virtual ~ CCiscoLineDevSpecificPortRegistrationPerCall () {}
   DWORD m_NumAffectedDevices;
   DWORD m_AffectedDeviceID[10];
   DWORD m_MediaCapCount;
   MEDIA_CAPSm_MediaCaps;
   virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
// subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_RECEIVE_RTP_INFO

DWORD m_NumAffectedDevices:

TSP returns this value. It contains the number of deviceIDs in the m_AffectedDeviceID array that are valid. Any device with multiple directory numbers that are assigned to it will have multiple TAPI lines, one per directory number.

DWORD m_AffectedDeviceID[10]:

TSP returns this value. It contains the list of deviceIDs for any device that is affected by this call. Do not call lineDevSpecific for any other device in this list.

DWORD m_mediaCapCount

The number of codecs that are supported for this line.

MEDIA_CAPS m_MediaCaps -

A data structure with the following format:

```
typedef struct {
DWORD MediaPayload;
DWORD MaxFramesPerPacket;
DWORD G723BitRate;
} MEDIA_CAPS[MAX_MEDIA_CAPS_PER_DEVICE];
```

This data structure defines each codec that is supported on a line. The limit specifies 18. The following description applies for each member in the MEDIA_CAPS data structure:

MediaPayload is an enumerated integer that contains one of the following values.

```
enum{
Media_Payload_G711Alaw64k = 2,
Media_Payload_G711Alaw56k = 3, // "restricted"
Media_Payload_G711Ulaw64k = 4,
Media_Payload_G711Ulaw56k = 5, // "restricted"
Media_Payload_G722_64k = 6,
Media_Payload_G722_56k = 7,
Media_Payload_G722_48k = 8,
Media_Payload_G7231 = 9,
Media_Payload_G728 = 10,
Media_Payload_G729 = 11,
Media_Payload_G729AnnexA = 12,
Media_Payload_G729AnnexB = 15,
Media_Payload_G729AnnexAwAnnexB = 16,
Media_Payload_GSM_Full_Rate = 18,
Media_Payload_GSM_Half_Rate = 19,
Media_Payload_GSM_Enhanced_Full_Rate = 20,
Media_Payload_Wide_Band_256k = 25,
Media_Payload_Data64 = 32,
Media_Payload_Data56 = 33,
Media_Payload_GSM = 80,
Media_Payload_G726_32K = 82,
Media_Payload_G726_24K = 83,
Media_Payload_G726_16K = 84,
// Media_Payload_G729_B = 85,
// Media_Payload_G729_B_LOW_COMPLEXITY = 86,
} Media_PayloadType;
```

MaxFramesPerPacket should read as MaxPacketSize and comprises a 16 bit integer that is specified in milliseconds. It indicates the RTP packet size. Typically, this value gets set to 20.

G723BitRate comprises a six byte field that contains either the G.723.1 information bit rate, or gets ignored. The values for the G.723.1 field comprises values that are enumerated as follows.

```
enum
{
Media_G723BRate_5_3 = 1, //5.3Kbps
```

```
Media_G723BRate_6_4 = 2 //6.4Kbps
} Media_G723BitRate;
```

Setting RTP Parameters for Call

The CCiscoLineDevSpecificSetRTPParamsForCall class sets the RTP parameters for a specific call.

Note This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificSetRTPParamsForCall
```

Class Details

```
class CciscoLineDevSpecificSetRTPParamsForCall: public CCiscoLineDevSpecific
{
public:
   CciscoLineDevSpecificSetRTPParamsForCall () :
CCiscoLineDevSpecific(SLDST_USER_SET_RTP_INFO) {}
   virtual ~ CciscoLineDevSpecificSetRTPParamsForCall () {}
   virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
   // subtract out the virtual function table pointer
   DWORD m_RecieveIP;   // UDP audio reception IP
   DWORD m_RecievePort; // UDP audio reception port
   };
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_SET_RTP_INFO

DWORD m_ReceiveIP

This specifies the RTP audio reception IP address in the network byte order to set for the call.

DWORD m_ReceivePort

This specifies the RTP audio reception port in the network byte order to set for the call.

Redirect Set Original Called ID

The CCiscoLineDevSpecificRedirectSetOrigCalled class redirects a call to another party while it sets the original called ID of the call to any other party.

Note This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificRedirectSetOrigCalled
```

Class Details

```
class CCiscoLineDevSpecificRedirectSetOrigCalled: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificRedirectSetOrigCalled () :
CCiscoLineDevSpecific(SLDST_REDIRECT_SET_ORIG_CALLED) {}
    virtual ~ CCiscoLineDevSpecificRedirectSetOrigCalled () {}
    char m_DestDirn[25];
    char m_SetOriginalCalledTo[25];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) -4) ;
}
```

Parameters

DWORD m_MsgType

Equals SLDST_REDIRECT_SET_ORIG_CALLED

char m_DestDirn[25]

Indicates the destination of the redirect. If this request is being used to transfer to voice mail, set this field to the voice mail pilot number of the DN of the line for the voice mail, to which you want to transfer.

char m_SetOriginalCalledTo[25]

Indicates the DN to which the OriginalCalledParty needs to be set. If this request is being used to transfer to voice mail, set this field to the DN of the line for the voice mail, to which you want to transfer.

HCALL hCall (in lineDevSpecific parameter list)

Equals the handle of the connected call.

Join

The CCiscoLineDevSpecificJoin class joins two or more calls into one conference call. Each call that is being joined can be in the ONHOLD or the CONNECTED call state.

The Cisco Unified Communications Manager may succeed in joining some calls that are specified in the Join request, but not all. In this case, the Join request will succeed and the Cisco Unified Communications Manager attempts to join as many calls as possible.

Note This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificJoin
```

Class Details

```
class CCiscoLineDevSpecificJoin : public CCiscoLineDevSpecific{
    public:
        CCiscoLineDevSpecificJoin () : CCiscoLineDevSpecific(SLDST_JOIN) {}
        virtual ~ CCiscoLineDevSpecificJoin () {}
        DWORD m_CallIDsToJoinCount;
        CALLIDS_TO_JOIN m_CallIDsToJoin;
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_JOIN

DWORD m_CallIDsToJoinCount

The number of callIDs that are contained in the m_CallIDsToJoin parameter.

CALLIDS_TO_JOIN m_CallIDsToJoin

A data structure that contains an array of dwCallIDs to join with the following format:

```
typedef struct {
    DWORD   CallID; // dwCallID to Join
} CALLIDS_TO_JOIN[MAX_CALLIDS_TO_JOIN];
```

where MAX_CALLIDS_TO_JOIN is defined as:

```
const DWORD MAX_CALLIDS_TO_JOIN = 14;
```

HCALL hCall (in LineDevSpecific parameter list)

Equals the handle of the call that is being joined with callIDsToJoin to create the conference.

Set User SRTP Algorithm IDs

The CciscoLineDevSpecificUserSetSRTPAlgorithmID class gets used to allow applications to set SRTP algorithm IDs. To use this class, ensure the lineNegotiateExtVersion API is called before opening the line. When calling lineNegotiateExtVersion, ensure the highest bit or second highest bit is set on both the dwExtLowVersion and dwExtHighVersion parameters. This causes the call to lineOpen to behave differently. The line does not actually opens, but waits for a lineDevSpecific call to complete the open with more information. Provide the extra information that is required in the CciscoLineDevSpecificUserSetSRTPAlgorithmID class.

Note This extension is only available if extension version 0x80070000, 0x4007000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificUserSetSRTPAlgorithmID
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened. (0x80070000 or 0x4007000 with the dwExtLowVersion and dwExtHighVersion parameters)
2. Call lineOpen for the deviceID of the line that is to be opened.
3. Call lineDevSpecific with a CciscoLineDevSpecificUserSetSRTPAlgorithmID message in the lpParams parameter to specify SRTP algorithm IDs.
4. Call lineDevSpecific with either CciscoLineDevSpecificPortRegistrationPerCall or CCiscoLineDevSpecificUserControlRTPStream message in the lpParams parameter.

Class Detail

```
class CciscoLineDevSpecificUserSetSRTPAlgorithmID: public CCiscoLineDevSpecific{
  public:
    CciscoLineDevSpecificUserSetSRTPAlgorithmID () :
    CCiscoLineDevSpecific(SLDST_USER_SET_SRTP_ALGORITHM_ID),
    m_SRTPAlgorithmCount(0),
    m_SRTP_Fixed_Element_Size(4)
{
}

    virtual ~ CciscoLineDevSpecificUserSetSRTPAlgorithmID () {}
      DWORD m_SRTPAlgorithmCount;     //Maximum is MAX_CISCO_SRTP_ALGORITHM_IDS
   DWORD m_SRTP_Fixed_Element_Size;//Should be size of DWORD, it should be always
 4.
      DWORD m_SRTPAlgorithm_Offset;   //offset from beginning of the message
buffer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out
the
virtual function table pointer
};
```

Supported Algorithm Constants

```
enum CiscoSRTPAlgorithmIDs{
    SRTP_NO_ENCRYPTION = 0,
    SRTP_AES_128_COUNTER = 1
};
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_SET_SRTP_ALGORITHM_ID

DWORD m_SRTPAlgorithmCount

This numbers of algorithm IDs that are specified in this message.

DWORD m_SRTP_Fixed_Element_Size

Should be size of DWORD, it should be always 4.

DWORD m_SRTPAlgorithm_Offset

Offset from the beginning of the message buffer. This is offset where you start put algorithm ID array.

Note Be aware that the dwSize should be recalculated based on size of the structure, m_SRTPAlgorithmCount and m_SRTP_Fixed_Element_Size.

Explicit Acquire

The CCiscoLineDevSpecificAcquire class gets used to explicitly acquire any CTI controllable device.

If a Superprovider application needs to open any CTI Controllable device on the Cisco Unified Communications Manager system, the application should explicitly acquire that device by using the above interface. After successful response, it can open the device as usual.

Note Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificAcquire
```

Class Details

```
class CCiscoLineDevSpecificAcquire : public CCiscoLineDevSpecific{
    public:
        CCiscoLineDevSpecificAcquire () : CCiscoLineDevSpecific(SLDST_ACQUIRE)
{}
        virtual ~ CCiscoLineDevSpecificAcquire () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_ACQUIRE

m_DeviceName[16]

The DeviceName that needs to be explicitly acquired.

Explicit De-Acquire

The CCiscoLineDevSpecificDeacquire class is used to explicitly de-acquire the explicitly acquired device.

If a Superprovider application has explicitly acquired any CTI Controllable device on the Cisco Unified Communications Manager system, then the application should explicitly De-acquire that device by using the above interface.

Note Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificDeacquire
```

Class Details

```
class CCiscoLineDevSpecificDeacquire : public CCiscoLineDevSpecific{
    public:
CCiscoLineDevSpecificDeacquire () : CCiscoLineDevSpecific(SLDST_ACQUIRE) {}
        virtual ~ CCiscoLineDevSpecificDeacquire () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_DEACQUIRE

char m_DeviceName[16]

The DeviceName that needs to be explicitly de-acquired.

Redirect FAC CMC

The CCiscoLineDevSpecificRedirectFACCMC class is used to redirect a call to another party that requires a FAC, CMC, or both.

Note Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificRedirectFACCMC
```

If the FAC is invalid, the TSP will return a new device-specific error code LINEERR_INVALIDFAC. If the CMC is invalid, the TSP will return a new device-specific error code LINEERR_INVALIDCMC.

Class Detail

```
class CCiscoLineDevSpecificRedirectFACCMC: public CCiscoLineDevSpecific{
public:
    CCiscoLineDevSpecificRedirectFACCMC () :
CCiscoLineDevSpecific(SLDST_REDIRECT_FAC_CMC) {}
    virtual ~ CCiscoLineDevSpecificRedirectFACCMC () {}
    char m_DestDirn[49];
    char m_FAC[17];
    char m_CMC[17];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) -4) ;
}
```

Parameters

DWORD m_MsgType

Equals SLDST_REDIRECT_FAC_CMC

char m_DestDirn[49]

Indicates the destination of the redirect.

char m_FAC[17]

Indicates the FAC digits. If the application does not want to pass any FAC digits, it must set this parameter to a NULL string.

char m_CMC[17]

Indicates the CMC digits. If the application does not want to pass any CMC digits, it must set this parameter to a NULL string.

HCALL hCall (in lineDevSpecific parameter list)

Equals the handle of the call to be redirected.

Blind Transfer FAC CMC

The CCiscoLineDevSpecificBlindTransferFACCMC class is used to blind transfer a call to another party that requires a FAC, CMC, or both. If the FAC is invalid, the TSP will return a new device specific error code LINEERR_INVALIDFAC. If the CMC is invalid, the TSP will return a new device specific error code LINEERR_INVALIDCMC.

Note Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificBlindTransferFACCMC
```

Class Detail

```
class CCiscoLineDevSpecificBlindTransferFACCMC: public CCiscoLineDevSpecific{
public:
    CCiscoLineDevSpecificBlindTransferFACCMC () :
CCiscoLineDevSpecific(SLDST_BLIND_TRANSFER_FAC_CMC) {}
    virtual ~ CCiscoLineDevSpecificBlindTransferFACCMC () {}
    char m_DestDirn[49];
    char m_FAC[17];
    char m_CMC[17];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) -4) ;
}
```

Parameters

DWORD m_MsgType

Equals SLDST_BLIND_TRANSFER_FAC_CMC

char m_DestDirn[49]

Indicates the destination of the blind transfer.

char m_FAC[17]

Indicates the FAC digits. If the application does not want to pass any FAC digits, it must set this parameter to a NULL string.

char m_CMC[17]

Indicates the CMC digits. If the application does not want to pass any CMC digits, it must set this parameter to a NULL string.

HCALL hCall (in lineDevSpecific parameter list)

Equals the handle of the call that is to be blind transferred.

CTI Port Third Party Monitor

The CCiscoLineDevSpecificCTIPortThirdPartyMonitor class is used for opening CTI ports in third-party mode.

To use this class, ensure the lineNegotiateExtVersion API is called before opening the line. When calling lineNegotiateExtVersion, ensure the highest bit is set on both the dwExtLowVersion and dwExtHighVersion parameters. This causes the call to lineOpen to behave differently. The line does not actually open, but waits for a lineDevSpecific call to complete the open with more information. Provide the extra information that is required in the CCiscoLineDevSpecificCTIPortThirdPartyMonitor class.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificCTIPortThirdPartyMonitor
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened. (OR 0x80000000 with the dwExtLowVersion and dwExtHighVersion parameters)

2. Call lineOpen for the deviceID of the line that is to be opened.
3. Call lineDevSpecific with a CCiscoLineDevSpecificCTIPortThirdPartyMonitor message in the lpParams parameter.

Note Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated.

Class Detail

```
class CCiscoLineDevSpecificCTIPortThirdPartyMonitor: public CCiscoLineDevSpecific
{
public:
   CCiscoLineDevSpecificCTIPortThirdPartyMonitor () :
   CCiscoLineDevSpecific(SLDST_CTI_PORT_THIRD_PARTY_MONITOR) {}
   virtual ~ CCiscoLineDevSpecificCTIPortThirdPartyMonitor () {}
   virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
// subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_CTI_PORT_THIRD_PARTY_MONITOR

Send Line Open

The CciscoLineDevSpecificSendLineOpen class is used for general delayed open purpose. To use this class, ensure the lineNegotiateExtVersion API is called before opening the line. When calling lineNegotiateExtVersion, ensure the second highest bit is set on both the dwExtLowVersion and dwExtHighVersion parameters. This causes the call to lineOpen to behave differently. The line does not actually open, but waits for a lineDevSpecific call to complete the open with more information. The extra information required is provided in the CciscoLineDevSpecificUserSetSRTPAlgorithmID class.

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificSendLineOpen
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened. (0x40070000 with the dwExtLowVersion and dwExtHighVersion parameters).
2. Call lineOpen for the deviceID of the line that is to be opened.
3. Call other lineDevSpecific, like CciscoLineDevSpecificUserSetSRTPAlgorithmID message in the lpParams parameter to specify SRTP algorithm IDs.
4. Call lineDevSpecific with either CciscoLineDevSpecificSendLineOpen to trigger the lineopen from TSP side.

Note Be aware that this extension is only available if extension version 0x40070000 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificSendLineOpen: public CCiscoLineDevSpecific
  {
  public:
    CciscoLineDevSpecificSendLineOpen () :
    CCiscoLineDevSpecific(SLDST_SEND_LINE_OPEN) {}

    virtual ~ CciscoLineDevSpecificSendLineOpen () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
 // subtract out the virtual function table pointer
};
```

Set Intercom SpeedDial

Use the CciscoLineSetIntercomSpeeddial class to allow application to set or reset SpeedDial/Label on an intercom line.

Note Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated

```
CCiscoLineDevSpecific
|
+--CciscoLineSetIntercomSpeeddial
```

Procedure

1. Call lineNegotiateExtVersion for the deviceID of the line that is to be opened (0x00080000 or higher).
2. Call lineOpen for the deviceID of the line that is to be opened.
3. Wait for line in service.
4. Call CciscoLineSetIntercomSpeeddial to set or reset speed dial setting on the intercom line.

Class Detail

```
class CciscoLineSetIntercomSpeeddial: public CCiscoLineDevSpecific  {
  public:
    CciscoLineSetIntercomSpeeddial () :
    CCiscoLineDevSpecific(SLDST_LINE_SET_INTERCOM_SPEEDDIAL) {}

    virtual ~ CciscoLineSetIntercomSpeeddial () {}
    DWORD SetOption;        //0 = clear app value, 1 = set App Value
    char Intercom_DN[MAX_DIRN];
    char Intercom_Ascii_Label[MAX_DIRN];
    wchar_t Intercom_Unicode_Label[MAX_DIRN];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
```

```
 // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_SET_INTERCOM_SPEEDDIAL

DWORD SetOption

Use this parameter to indicate whether the application wants to set a new intercom speed dial value or clear the previous value. 0 = clear, 1 = set.

Char Intercom_DN [MAX_DIRN]

A DN array that indicates the intercom target

Char Intercom_Ascii_Label[MAX_DIRN]

Indicates the ASCII value of the intercom line label

Wchar_tIntercom_Unicode_Label[MAX_DIRN]

Indicates the Unicode value of the intercom line label

MAX_DIRN is defined as 25.

Intercom Talk Back

Use the CciscoLineIntercomTalkback class to allow the application to initiate talk back on an incoming intercom call on an intercom line.

Note Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CciscoLineIntercomTalkback
```

Class Detail

```
class CciscoLineIntercomTalkback: public CCiscoLineDevSpecific{
  public:
    CCciscoLineIntercomTalkback () :
    CCiscoLineDevSpecific(SLDST_INTERCOM_TALKBACK) {}

    virtual ~ CciscoLineIntercomTalkback () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
 // subtract out the virtual function table pointer
};
```

Redirect with Feature Priority

CciscoLineRedirectWithFeaturePriority enables an application to redirect calls with specified feature priorities. The following is the structure of CciscoLineDevSpecific:

```
CCiscoLineDevSpecific
|
+--CciscoLineRedirectWithFeaturePriority
```

Note Be aware that this extension is only available if the extension version 0x00080001 or higher is negotiated.

Detail

```
class CciscoLineRedirectWithFeaturePriority: public CCiscoLineDevSpecific  {
  public:
    CciscoLineRedirectWithFeaturePriority() :
    CCiscoLineDevSpecific(SLDST_REDIRECT_WITH_FEATURE_PRIORITY) {}

    virtual ~ CciscoLineRedirectWithFeaturePriority () {}
    CiscoDoNotDisturbFeaturePriority FeaturePriority;
    char m_DestDirn[25];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
 // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_REDIRECT_WITH_FEATURE_PRIORITY

enum CiscoDoNotDisturbFeaturePriority {CallPriority_NORMAL = 1, CallPriority_URGENT = 2, CallPriority_EMERGENCY = 3};

This identifies the priorities.

char m_DestDirn[25];

This is redirect destination.

Start Call Monitoring

Use CCiscoLineDevSpecificStartCallMonitoring to allow application to send a start monitoring request for the active call on a line.

Note Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificStartCallMonitoring
```

Class Detail

```
class CCiscoLineDevSpecificStartCallMonitoring: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificStartCallMonitoring () :
      CCiscoLineDevSpecific(SLDST_START_CALL_MONITORING) {}
    virtual ~    CCiscoLineDevSpecificStartCallMonitoring () {}
    DWORD m_PermanentLineID ;
    DWORD m_MonitorMode;
    DWORD m_ToneDirection;
   // subtract out the virtual function table pointer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
} ;
```

Parameters

DWORD m_MsgType

Equals SLDST_START_MONITORING

DWORD m_PermanentLineID

The permanent lineID of the line whose active call has to be monitored.

DWORD MonitorMode

This can have the following enum value:

```
enum             {
        MonitorMode_None    = 0,
        MonitorMode_Silent  = 1,
        MonitorMode_Whisper = 2,   //  Not used
        MonitorMode_Active  = 3    //  Not used
    } MonitorMode;
```

Note Silent Monitoring mode represents the only mode that is supported in which the supervisor cannot talk to the agent.

DWORD PlayToneDirection

This parameter specifies whether a tone should play at the agent or customer phone when monitoring starts. It can have following enum values:

```
enum
     {
     PlayToneDirection_LocalOnly = 0,
     PlayToneDirection_RemoteOnly,
     PlayToneDirection_BothLocalAndRemote,
     PlayToneDirection_NoLocalOrRemote
    } PlayToneDirection
```

Return Values

```
-LINERR_OPERATIONFAILED-LINEERR_OPERATIONUNAVAIL
-LINEERR_RESOURCEUNAVAIL
-LINEERR_BIB_RESOURCE_UNAVAIL
-LINERR_PENDING_REQUEST
-LINEERR_OPERATION_ALREADY_INPROGRESS
-LINEERR_ALREADY_IN_REQUESTED_STATE
-LINEERR_PRIMARY_CALL_INVALID
-LINEERR_PRIMARY_CALL_STATE_INVALID
```

Start Call Recording

Use CCiscoLineDevSpecificStartCallRecording to allow applications to send a recording request for the active call on that line.

Note Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificStartCallRecording
```

Class Detail

```
class CCiscoLineDevSpecificStartCallRecording: public CCiscoLineDevSpecific{
public:
CCiscoLineDevSpecificStartCallRecording () :
CCiscoLineDevSpecific(SLDST_START_CALL_RECORDING) {}

virtual ~CCiscoLineDevSpecificStartCallRecording () {}

DWORD m_ToneDirection;
DWORD m_InvocationType;

virtual DWORD dwSize(void) const {
// subtract out the virtual function table pointer
return sizeof(*this)-sizeof(void*);
}
};
```

Parameters

DWORD m_MsgType

Equals SLDST_START_RECORDING

DWORD m_ToneDirection

This parameter specifies whether a tone should play at the agent or customer phone when recording starts. It can have the following values:

```
enum     {
     PlayToneDirection_NoLocalOrRemote = 0,
     PlayToneDirection_LocalOnly,
     PlayToneDirection_RemoteOnly,
     PlayToneDirection_BothLocalAndRemote,
 PlayToneDirection_NotApplicable
    } PlayToneDirection
```

DWORD m_InvocationType

This parameter specifies whether the recording status is displayed on the phone (user-controlled recording) or not displayed (silent recording).

```
enum RecordingInvocationType    {
    RecordingInvocationType_SilentRecording = 1,
    RecordingInvocationType_UserControlledRecording = 2
    }
```

Return Values

```
-LINERR_OPERATIONFAILED-LINEERR_OPERATIONUNAVAIL
-LINEERR_INVALCALLHANDLE
-LINEERR_BIB_RESOURCE_UNAVAIL
-LINERR_PENDING_REQUEST
-LINERR_OPERATION_ALREADY_INPROGRESS
-LINEERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING
-LINEERR_RECORDING_CONFIG_NOT_MATCHING
```

StopCall Recording

Use CCiscoLineDevSpecificStopCallRecording to allow application to stop recording a call on that line.

Note Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificStopCallRecording
```

Class Detail

```
class CCiscoLineDevSpecificStopCallRecording: public CCiscoLineDevSpecific
{
public:
   CCiscoLineDevSpecificStopCallRecording () :
   CCiscoLineDevSpecific(SLDST_STOP_CALL_RECORDING) {}

   virtual ~CCiscoLineDevSpecificStopCallRecording () {}

   DWORD m_InvocationType;

   virtual DWORD dwSize(void) const {
   // subtract out the virtual function table pointer
```

```
    return sizeof(*this)-sizeof(void*);
    }

} ;
```

Parameters

DWORD m_MsgType

Equals SLDST_STOP_RECORDING

DWORD m_InvocationType

This parameter specifies whether the recording status is displayed on the phone (user-controlled recording) or not displayed (silent recording).

```
enum RecordingInvocationType
    {
    RecordingInvocationType_SilentRecording = 1,
    RecordingInvocationType_UserControlledRecording = 2
    }
```

Return Values

```
-LINERR_OPERATIONFAILED-LINEERR_OPERATIONUNAVAIL
-LINEERR_INVALCALLHANDLE
-LINERR_PENDING_REQUEST
-LINEERR_RECORDING_INVOCATION_TYPE_NOT_MATCHING
-LINEERR_NO_RECORDING_SESSION
-LINEERR_RECORDING_SESSION_INACTIVE
```

Set IPv6 Address and Mode

Use the CciscoLineDevSpecificSetIPv6AddressAndMode class to allow the application to set IPv6 address and addressing mode during the static registration. To use this class, ensure the lineNegotiateExtVersion API must be called before opening the line. When calling lineNegotiateExtVersion, ensure the highest bit or the second highest must be set on both the dwExtLowVersion and dwExtHighVersion parameters. This causes the call to lineOpen to behave differently. The line does not actually open, but waits for a lineDevSpecific call to complete the open with more information. The extra information required is provided in the CciscoLineDevSpecificSetIPv6AddressAndMode class.

```
CCiscoLineDevSpecific|
+--CciscoLineDevSpecificSetIPv6AddressAndMode
```

Note This extension is available only if extension version 0x80090000, 0x40090000 or higher is negotiated.

Procedure

1. Open Call lineNegotiateExtVersion for the deviceID of the line (0x80090000 or 0x40090000 with the dwExtLowVersion and dwExtHighVersion parameters)

2. Open Call lineOpen for the deviceID of the line.
3. Call lineDevSpecific with a CciscoLineDevSpecificSetIPv6AddressAndMode message in the lpParams parameter to specify IPv6 address and the IP Addressing mode as IPv6.

Class Detail

```
class CciscoLineDevSpecificSetIPv6AddressAndMode: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificSetIPv6AddressAndMode() :
        CCiscoLineDevSpecific(SLDST_USER_SET_IPv6_ADDRESS_AND_MODE),
           m_ReceivePort(-1), m_IPAddressMode( (IpAddressingMode) 1)
    {
    }
    virtual ~ CciscoLineDevSpecificSetIPv6AddressAndMode()
    {
    }
    char m_ReceiveIPv6Address[16];
    DWORD m_ReceivePort;
    IpAddressingMode m_IPAddressMode;
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) -4;
    } // subtract out the virtual function table pointer
};
```

Parameters

DWORDm_MsgType

Equals SLDST_USER_SET_IPv6_ADDRESS

Charm_ReceiveIPv6Address[16]

User has to specify the IPv6 address to register the CTI Port with

DWORDm_ReceivePort

This specifies the port number for the user to register the CTI Port.

Intm_IPAddressMode

This specifies the Addressing mode with which user wants the CTI Port/RP registered.

Set RTP Parameters for IPv6 Calls

Use CciscoLineDevSpecificSetRTPParamsForCallIPv6 class to set the RTP parameters for calls for which you must specify IPv6 address.

Note Be aware that this extension is available only if extension version 0x00090000 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificSetRTPParamsForCallIPv6: public CCiscoLineDevSpecific
{
public:
   CciscoLineDevSpecificSetRTPParamsForCallIPv6 () :
   CCiscoLineDevSpecific(SLDST_USER_SET_RTP_INFO_IPv6) {}
   virtual ~ CciscoLineDevSpecificSetRTPParamsForCallIPv6 () {}
   virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
 // subtract out the virtual function table pointer
   char m_RecieveIPv6[16];   // UDP audio reception IPv6
   DWORD m_RecievePort // UDP audio reception port
  };
```

Parameters

DWORD m_MsgType

Equals SLDST_USER_SET_RTP_INFO_IPv6

DWORD m_ReceiveIPv6

This is the RTP audio reception IPv6 address to set for the call

DWORD m_RecievePort

This is the RTP audio reception port to set for the call.

Direct Transfer

Use the CciscoLineDevSpecificDirectTransfer to transfer calls across lines or on the same line.

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificDirectTransfer
```

Note Be aware that this extension is available only if extension version 0x00090001 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificDirectTransfer: public CCiscoLineDevSpecific{
    public:
        CciscoLineDevSpecificDirectTransfer () :
           CCiscoLineDevSpecific(SLDST_DIRECT_TRANSFER) {}
        virtual ~ CciscoLineDevSpecificDirectTransfer () {}
        DWORD m_CallIDsToTransfer;
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_DIRECT_TRANSFER

DWORD m_CallIDsToTransfer

Consult dwCallID to be transferred

HCALL hCall (in LineDevSpecific parameter list)

Equals the handle of the call that is being transferred.

RegisterCallPickUpGroupForNotification

The CciscoLineDevSpecificRegisterCallPickupGroupForNotification class is used to register the call Pickup Group for notification on calls for Pickup.

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificRegisterCallPickupGroupForNotification
```

Note This extension is available only if extension version 0x000A0000 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificRegisterCallPickupGroupForNotification:
  public CCiscoLineDevSpecific
{
    public:
        CciscoLineDevSpecificRegisterCallPickupGroupForNotification ():
CCiscoLineDevSpecific (SLDST_CALLPICKUP_GROUP_REGISTER) {}
   virtual ~ CciscoLineDevSpecificRegisterCallPickupGroupForNotification () {}
   char callPickupGroupDN[MAX_DIRN];
   char callPickupGroupPartition[MAX_PARTITION];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_CALLPICKUP_GROUP_REGISTER

Char CallPickupGroupDN []

-DN of the pickup Group

Char CallPickupGroupPartition []

-Partition of the PickupGroup

UnRegisterCallPickUpGroupForNotification

The CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification class is used to unregister the call Pickup Group for notification on calls for Pickup.

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification
```

Note This extension is available only if extension version 0x000A0000 or higher is negotiated

Class Details

```
class CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification:
 public CCiscoLineDevSpecific{
    Public:
        CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification () :
 CCiscoLineDevSpecific(SLDST_CALLPICKUP_GROUP_UNREGISTER) {}
        virtual ~ CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification
() {}
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_CALLPICKUP_GROUP_UNREGISTER

CallPickUpRequest

The CciscoLineDevSpecificCallPickupRequest class is used to Pickup the call from the PickGroup.

```
CCiscoLineDevSpecific

+--CciscoLineDevSpecificCallPickupRequest
```

Note This extension is available only if extension version 0x000A0000 or higher is negotiated.

Class Details

```
class CciscoLineDevSpecificCallPickupRequest:
 public CCiscoLineDevSpecific{
    public:
        CciscoLineDevSpecificCallPickupRequest ():
   CCiscoLineDevSpecific (SLDST_CALLPICKUP_CALL) {}
```

```
        virtual ~ CciscoLineDevSpecificCallPickupRequest () {}
        DWORD PickupType;
        char PickupGroupDN[MAX_DIRN];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_CALLPICKUP_CALL

Char PickupGroupDN []

-DN of the pickup Group/DN;will be required for GroupCallPickUp and DirectedCallPickUp

DWORD PickupType

-indicates the type of pickup (CtiCallPickUp, CtiGroupCallPickUp, , CtiOtherPickup, DirectedCallPickup)

```
enum CallPickupType{
        CallPickup_Simple = 0,
        CallPickup_Group  = 1,
        CallPickup_Other  = 2,
        CallPickup_Direct = 3
};
```

Start Send Media to BIB

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificStartSendMediaToBIBRequest
```

Description

The CCiscoLineDevSpecificStartSendMediaToBIBRequest class allows the application to initiate agent greeting to the customer call.

Note This extension is only available if extension version 0x000B0000 or higher is negotiated.

TAPI line handle and TAPI call handle are required for this request.

Class Detail

```
class CCiscoLineDevSpecificStartSendMediaToBIBRequest:
 public CCiscoLineDevSpecific{
    public:
       CCiscoLineDevSpecificStartSendMediaToBIBRequest (): CCiscoLineDevSpecific
 (SLDST_START_SEND_MEDIA_TO_BIB) {}
        virtual ~ CCiscoLineDevSpecificStartSendMediaToBIBRequest () {}
      char m_IVRDN [49];
```

```
      char m_CGPNTOIVR [49];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
```

Parameters

DWORD m_MsgType

equals SLDST_START_SEND_MEDIA_TO_BIB

char m_IVRDN [49]

IVR port DN where Agent Greeting will be played from

char m_CGPNTOIVR [49]

The CallingPartyDN passed to IVR. The application can use this field to pass DN as CallingPartyDN for the agent greeting call.

Stop Send Media to BIB

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificStopSendMediaToBIBRequest
```

Description

The CCiscoLineDevSpecificStopSendMediaToBIBRequest class allows the application to stop the agent greeting that is playing on the agent-to-customer call.

Note This extension is only available if extension version 0x000B0000 or higher is negotiated.

TAPI line handle and TAPI call handle are required for this request.

Class Detail

```
class CCiscoLineDevSpecificStopSendMediaToBIBRequest:
 public CCiscoLineDevSpecific
{
    public:
        CCiscoLineDevSpecificStopSendMediaToBIBRequest ():
CCiscoLineDevSpecific (SLDST_STOP_SEND_MEDIA_TO_BIB) {}
        virtual ~ CCiscoLineDevSpecificStopSendMediaToBIBRequest () {}
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_START_SEND_MEDIA_TO_BIB

Agent Zip Tone

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificEnableFeatureSupport
```

Description

The CCiscoLineDevSpecificPlaytone class is used to play the tone (Zip Tone) to the direction specified in the request.

Note This extension is only available if extension version 0x000B0000 or higher is negotiated.

Class Detail

```
class CCiscoLineDevSpecificPlaytone:
public CCiscoLineDevSpecific //AgentZiptone
{
  public:
    CCiscoLineDevSpecificPlaytone() :
    CCiscoLineDevSpecific(SLDST_PLAY_TONE)
    {
    }
    virtual ~ CCiscoLineDevSpecificPlaytone()
    {
    }
    DWORD m_Tone;
    DWORD m_ToneDirection;
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) -4;
    } // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_Tone -Indicates the Tone type

equals CTONE_ZIP

DWORD m_ToneDirection -Indicates the direction of the tone to be played;

equals Tonedirection (Local/Remote)

Early Offer

New error Code – LINEERR_REGISTER_GETPORT_SUPPORT_MISMATCH 0xC000000F

Enable Feature

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificEnableFeatureSupport
```

Description

The CciscoLineDevSpecificEnableFeatureSupport class allows application to enhance or update feature support.

Note This extension is only available if extension version 0x000B0000 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificEnableFeatureSupport:
 public CCiscoLineDevSpecific{
  public:
    CciscoLineDevSpecificEnableFeatureSupport() :
        CCiscoLineDevSpecific(SLDST_ENABLE_FEATURE_SUPPORT)
    {
    }

    virtual ~ CciscoLineDevSpecificEnableFeatureSupport()
    {
    }
    DWORD m_Feature;
    DWORD m_Feature_Capability;
//Should have Value supported for Feature specified in m_Feature
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) -4;
    } // subtract out the virtual function table pointer
```

Parameters

DWORD m_MsgType

equals SLDST_ENABLE_FEATURE_SUPPORT

DWORD m_Feature

Feature value for which the capability needs to be changed and should have a value from the following Enum:

```
enum TspFeatureSupport
{
    Feature_unknown =    0,
    Feature_EarlyOffer =  1
};
```

DWORD m_Feature_Capability

The Capability information that needs to be changed/updated for the feature. This information changes depending on the feature.

Early Offer (Get Port) Support:

m_Feature should be Feature_EarlyOffer(1) and

m_Feature_Capability should have a value from following Enum:

```
enum TspFeatureOption{
    Feature_Disable =  0,
    Feature_Enable = 1
};
```

Sample Code:

Here is a sample code that illustrates how applications must use this devspecific type, and fill the Class Object to enable/disable the Early Offer feature support.

```
void main(){
… … … ….
CciscoLineDevSpecificEnableFeatureSupport featureObject;
featureObject.m_MsgType = SLDST_ENABLE_FEATURE_SUPPORT;
featureObject.m_Feature = Feature_EarlyOffer(1);
featureObject. m_Feature_Capability = Feature_Enable(1)/ Feature_Disable(0);

int result = TSPI_lineDevSpecific(dwRequestID,hdLine,
dwAddressID, NULL, &featureObject,
sizeof(CciscoLineDevSpecificEnableFeatureSupport));
… … … ….
… … … ….
}
New CiscoLineDevStateOutOfServiceReason_EMLogin and
CiscoLineDevStateOutOfServiceReason_EMLogout values in the
CiscoLineDevStateOtherReason enumeration type in CiscoLineDevSpecificMsg.h:

enum CiscoLineDevStateOutOfServiceReason
{
    CiscoLineDevStateOutOfServiceReason_Unknown = 0x00000000,
    CiscoLineDevStateOutOfServiceReason_CallManagerFailure = 0x00000001,
   CiscoLineDevStateOutOfServiceReason_ReHomeToHigherPriorityCM = 0x00000002,

    CiscoLineDevStateOutOfServiceReason_NoCallManagerAvailable = 0x00000003,
    CiscoLineDevStateOutOfServiceReason_DeviceFailure = 0x00000004,
    CiscoLineDevStateOutOfServiceReason_DeviceUnregistered = 0x00000005,
   CiscoLineDevStateOutOfServiceReason_EnergyWisePowerSavePlus = 0x00000006,

    CiscoLineDevStateOutOfServiceReason_EMLogin = 0x00000007,
    CiscoLineDevStateOutOfServiceReason_EMLogout =       0x00000008,
    CiscoLineDevStateOutOfServiceReason_CtiLinkFailure = 0x00000101
};
New CiscoLineDevStateCloseReason enumeration type in CiscoLineDevSpecificMsg.h:
enum CiscoLineDevStateCloseReason
{
    CiscoLineDevStateCloseReason_Unknown = 0,
    CiscoLineDevStateCloseReason_EMLogin = 1,
    CiscoLineDevStateCloseReason_EMLogout = 2
};

New CiscoLineDevStateOtherReason enumeration type in CiscoLineDevSpecificMsg.h:
```

```
enum CiscoLineDevStateOtherReason
{
CiscoLineDevStateOtherReason_Unknown = 0,
CiscoLineDevStateOtherReason_OtherLineInactive = 1,
CiscoLineDevStateOtherReason_OtherLineActive = 2,
CiscoLineDevStateOtherReason_OtherLineCapsChange = 3
};

New LINEERR_DEVICE_INACTIVE error is returned if an operation is invoked
on a line device in “inactive” state.
```

UpdateMonitorMode

```
CCiscoLineDevSpecific
|
+--CciscoLineDevSpecificMonitoringUpdateMode
```

Description

The CciscoLineDevSpecificMonitoringUpdateMode class allows the application to toggle between the silent monitoring and whisper coaching modes, and vice versa.

Note This extension is only available if extension version 0x000B0000 or higher is negotiated.

Class Detail

```
class CciscoLineDevSpecificMonitoringUpdateMode :
 public CCiscoLineDevSpecific
{
    public:
        CciscoLineDevSpecificMonitoringUpdateMode ():
CCiscoLineDevSpecific (SLDST_UPDATE_MONITOR_MODE) {}
        virtual ~ CciscoLineDevSpecificMonitoringUpdateMode () {}
  DWORD m_MonitorMode;
  DWORD m_ActiveToneDirection;
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

equals SLDST_UPDATE_MONITOR_MODE

DWORD m_MonitorMode

Monitoring mode to toggle to

DWORD m_ActiveToneDirection

Direction of the tone to be played

Add Remote Destination

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificAddRemoteDestination
```

Description

The CciscoLineDevSpecificAddRemoteDestination class is used to add new Remote Destination to CTI Remote Device.

Note This extension is only available on CTI Remote Device Line and if extension version 0x000C0000 or higher is negotiated.

Class Details

```
class CciscoLineDevSpecificAddRemoteDestination: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificAddRemoteDestination() :
        CCiscoLineDevSpecific(SLDST_ADD_REMOTE_DESTINATION)
    {
    }

virtual ~ CciscoLineDevSpecificAddRemoteDestination()
    {
    }

    char    m_RDNumber [MAX_CTI_LINE_DIR_SIZE];
    wchar_t m_UnicodeRDName [MAX_CTI_RD_UNICODE_DISPLAY_STRING];
    DWORD   m_ActiveRD;

virtual DWORD dwSize(void) const
    {
        returnsizeof(*this) - sizeof(void*);
    } // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_ADD_REMOTE_DESTINATION

char m_RDNumber [MAX_CTI_LINE_DIR_SIZE]

Remote Destination Number [*Mandatory Field]

wchar_t m_UnicodeRDName [MAX_UNICODE_DISPLAY_STRING]

unicode Remote Destination Name

DWORD m_activeRD

0 – if this Remote Destination is not Active

1 or greater – when this Remote Destination need to be set as Active Remote Destination

Remove Remote Destination

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificRemoveRemoteDestination
```

Description

The CciscoLineDevSpecificRemoveRemoteDestination class is used to remove Remote Destination from List of Remote Destinations of CTI Remote Device.

Note This extension is only available on CTI Remote Device Line and if extension version 0x000C0000 or higher is negotiated.

Class Details

```
class CciscoLineDevSpecificRemoveRemoteDestination: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificRemoveRemoteDestination() :
        CCiscoLineDevSpecific(SLDST_REMOVE_REMOTE_DESTINATION)
    {
    }

virtual ~ CciscoLineDevSpecificRemoveRemoteDestination()
    {
    }

    char    m_RDNumber [MAX_CTI_LINE_DIR_SIZE];

virtual DWORD dwSize(void) const
    {
        returnsizeof(*this) - sizeof(void*);
    } // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_REMOVE_REMOTE_DESTINATION

char m_RDNumber [MAX_CTI_LINE_DIR_SIZE]

Remote Destination Number [*Mandatory Field]

Note Remote Destination can be removed using Remote Destination Number which is used a unique key for Remote Destinations on a CTI Remote Device.

Update Remote Destination

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificUpdateRemoteDestination
```

Description

The CciscoLineDevSpecificUpdateRemoteDestination class is used to update Remote Destination information on a CTI Remote Device.

Note This extension is only available on CTI Remote Device Line and if extension version 0x000C0000 or higher is negotiated.

Class Details

```
class CciscoLineDevSpecificUpdateRemoteDestination: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificUpdateRemoteDestination() :
        CCiscoLineDevSpecific(SLDST_UPDATE_REMOTE_DESTINATION)
    {
    }

virtual ~ CciscoLineDevSpecificUpdateRemoteDestination()
    {
    }

    char    m_RDNumber [MAX_CTI_LINE_DIR_SIZE];
    wchar_t m_UnicodeRDName [MAX_CTI_RD_UNICODE_DISPLAY_STRING];
    char    m_NewRDNumber [MAX_CTI_LINE_DIR_SIZE];
    DWORD   m_ActiveRD;

virtual DWORD dwSize(void) const
    {
        returnsizeof(*this) - sizeof(void*);
    } // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals SLDST_UPDATE_REMOTE_DESTINATION.

char m_RDNumber [MAX_CTI_LINE_DIR_SIZE]

Current Remote Destination Number which need to be updated [*Mandatory Field]

wchar_t m_UnicodeRDName [MAX_UNICODE_DISPLAY_STRING]

unicode Remote Destination Name

char m_NewRDNumber [MAX_CTI_LINE_DIR_SIZE]

New Remote Destination Number [*Mandatory Field]

DWORD m_activeRD

0 – if this Remote Destination is not Active

1 or greater – when this Remote Destination need to be set as Active Remote Destination

lineHold Enhancement

The CciscoLineDevSpecificHoldEx class is used to put a call on hold and specifies media content that is played while a call is on hold.

Message Details

```
class CciscoLineDevSpecificHoldEx: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificHoldEx() :
        CCiscoLineDevSpecific(SLDST_HOLD_EX) {}
    virtual ~ CciscoLineDevSpecificHoldEx() {}
    char    m_MediaContentId [MAX_CISCO_TSP_MEDIA_CONTENT_ID_SIZE];
    DWORD   m_MediaContentIdLength;
    virtual DWORD dwSize(void) const {return sizeof(*this) - sizeof(void*);}
};
```

Parameters

char m_MediaContentId [MAX_CISCO_TSP_MEDIA_CONTENT_ID_SIZE]

String that represents media content identifier (mediaContentID). Only known to UCM (uploaded to UCM) media content can be played.

DWORD m_MediaContentIdLength

Actual length of the string in the m_MediaContentId field.

Cisco Line Device Feature Extensions

CCiscoLineDevSpecificFeature represents the parent class. Currently, it consist of only one subclass: CCiscoLineDevSpecificFeature_DoNotDisturb, which allows applications to enable and disable the Do-Not-Disturb feature on a device.

This following sections describe the line device feature-specific extensions to the TAPI structures that Cisco TSP supports:

- CCiscoLineDevSpecificFeature, on page 385
- Do-Not-Disturb, on page 387
- Do-Not-Disturb Change Notification Event, on page 387

CCiscoLineDevSpecificFeature

This section provides information on how to invoke Cisco-specific TAPI extensions with the CCiscoLineDevSpecificFeature class, which represents the parent class to all the following classes.

Note Be aware that this virtual class is provided for informational purposes only.

```
CCiscoLineDevSpecificFeature
```

Header File

The file CiscoLineDevSpecific.h contains the corresponding constant, structure, and class definitions for the Cisco lineDevSpecificFeature extension classes.

Class Detail

```
class CCiscoLineDevSpecificFeature
{
public:
  CCicsoLineDevSpecificFeature(const DWORD msgType): m_MsgType(msgType) {;}
  virtual ~ CCicsoLineDevSpecificFeature() {;}
  DWORD GetMsgType(void) const {return m_MsgType;}
  void* lpParams(void) const {return (void*)&m_MsgType;}
  virtual DWORD dwSize(void) const = 0;
private:
  DWORD m_MsgType;
};
```

Functions

lpParms()

Function that can be used to obtain a pointer to the parameter block

dwSize()

Function that returns size of the parameter block area

Parameter

m_MsgType

Specifies the type of message. The parameter value uniquely identifies the feature to invoke on the device. The PHONEBUTTONFUNCTION_ TAPI_Constants definition lists the valid feature identifiers. Currently, the only recognized value specifies PHONEBUTTONFUNCTION_DONOTDISTURB (0x0000001A).

Each subclass of CCiscoLineDevSpecificFeature includes a unique value that is assigned to the m_MsgType parameter.

Subclasses

Each subclass of CCiscoLineDevSpecificFeature carries a unique value that is assigned to the m_MsgType parameter. If you are using C instead of C++, this represents the first parameter in the structure.

Do-Not-Disturb

Use the CCiscoLineDevSpecificFeature_DoNotDisturb class in conjunction with the request to enable or disable the DND feature on a device.

The Do-Not-Disturb feature gives phone users the ability to go into a Do Not Disturb (DND) state on the phone when they are away from their phones or simply do not want to answer the incoming calls. A phone softkey, DND, allows users to enable or disable this feature.

```
CCiscoLineDevSpecificFeature
|
+--CCiscoLineDevSpecificFeature_DoNotDisturb
```

Class Detail

```
class CCiscoLineDevSpecificFeature_DoNotDisturb :
 public CCiscoLineDevSpecificFeature
{
public:
  CCiscoLineDevSpecificFeature_DoNotDisturb()
: CCiscoLineDevSpecificFeature(PHONEBUTTONFUNCTION_DONOTDISTURB),
 m_Operation((CiscoDoNotDisturbOperation)0) {}
virtual ~CCiscoLineDevSpecificFeature_DoNotDisturb() {}
virtual DWORD dwSize(void) const {return sizeof(*this)-4;}

CiscoDoNotDisturbOperation  m_Operation;
};
```

Parameters

DWORD m_MsgType

Equals PHONEBUTTONFUNCTION_DONOTDISTURB.

CiscoDoNotDisturbOperation m_Operation

Specifies a requested operation and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOperation {
    DoNotDisturbOperation_ENABLE    = 1,
    DoNotDisturbOperation_DISABLE   = 2
};
```

Do-Not-Disturb Change Notification Event

Cisco TSP notifies applications via the LINE_DEVSPECIFICFEATURE message about changes in the DND configuration or status. To receive change notifications, an application needs to enable the DEVSPECIFIC_DONOTDISTURB_CHANGED message flag with a lineDevSpecific SLDST_SET_STATUS_MESSAGES request.

The LINE_DEVSPECIFICFEATURE message notifies the application about device-specific events that occur on a line device. In the case of a Do-Not-Disturb Change Notification, the message includes information about the type of change that occurred on a device and the resulting feature status or configured option.

Message Details

```
LINE_DEVSPECIFICFEATUREdwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) PHONEBUTTONFUNCTION_DONOTDISTURB;
dwParam2 = (DWORD) typeOfChange;
dwParam3 = (DWORD) currentValue;

enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE      = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT    = 2
};

enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN  = 0,
    DoNotDisturbStatus_ENABLED  = 1,
    DoNotDisturbStatus_DISABLED = 2
};

enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED  = 1,
    DoNotDisturb_OPTION_CHANGED  = 2
};
```

Parameters

dwDevice

A handle to a line device

dwCallbackInstance

The callback instance that is supplied when the line is opened

dwParam1

Always equal to PHONEBUTTONFUNCTION_DONOTDISTURB for the Do-Not-Disturb change notification

dwParam2

Indicates type of change and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED  = 1,
    DoNotDisturb_OPTION_CHANGED  = 2
};
```

dwParam3

If the dwParm2 indicates status change with the value DoNotDisturb_STATUS_CHANGED, this parameter can comprise one of the following enum values:

```
enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN  = 0,
    DoNotDisturbStatus_ENABLED  = 1,
    DoNotDisturbStatus_DISABLED = 2
};
```

If the dwParm2 indicates option change with the value DoNotDisturb_OPTION_CHANGED, this parameter can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE      = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT    = 2
};
```

Cisco Phone Device-Specific Extensions

The following table lists and describes the subclasses of CiscoPhoneDevSpecific.

Table 22: Cisco Phone Device-Specific TAPI Functions

Cisco Functions	Synopsis
CCiscoPhoneDevSpecific, on page 389	The CCiscoPhoneDevSpecific class represents the parent class to the following classes.
Device Data PassThrough, on page 391	Allows the application to send the Device Specific XSI data through CTI.
Explicit Acquire, on page 360	Allows the application to acquire any CTI-controllable device that can get opened later in superprovider mode.
Explicit De-Acquire, on page 360	Allows the application to deacquire a CTI-controllable device that was explicitly acquired.
Request Call RTP Snapshot, on page 395	Allows the application to request secure RTP indicator for calls on the device.
Set Status Msgs, on page 392	Allows the application to set status bit map to enable specific DEVICE_DEVSPECIFIC messages to be sent to the application.
Set Unicode Display, on page 393	Sets the Unicode display on the phone.

CCiscoPhoneDevSpecific

This section provides information on how to perform Cisco TAPI-specific functions with the CCiscoPhoneDevSpecific class, which represents the parent class to all the following classes.

Note Be aware that this virtual class is provided for informational purposes only.

```
CCiscoPhoneDevSpecific|
+--CCiscoPhoneDevSpecificDataPassThrough
|
+--CCiscoPhoneDevSpecificSetStatusMsgs
```

```
|
+--CCiscoPhoneDevSpecificSetUnicodeDisplay
|
+--CCiscoPhoneDevSpecificAcquire
|
+--CCiscoPhoneDevSpecificDeacquire
|
+--CCiscoPhoneDevSpecificGetRTPSnapshot
```

Header File

The file CiscoLineDevSpecific.h contains the constant, structure, and class definition for the Cisco phone device-specific classes.

Class Detail

```
class CCiscoPhoneDevSpecific
{
public :
   CCiscoPhoneDevSpecific(DWORD msgType):m_MsgType(msgType) {;}
   virtual ~CCiscoPhoneDevSpecific() {;}
   DWORD GetMsgType (void) const { return m_MsgType;}
   void *lpParams(void) const {return (void*)&m_MsgType;}
   virtual DWORD dwSize(void) const = 0;
private :
   DWORD m_MsgType ;
}
```

Functions

lpParms()

Function that can be used to obtain the pointer to the parameter block

dwSize()

Function that will give the size of the parameter block area

Parameter

m_MsgType

Specifies the type of message.

Subclasses

Each subclass of CCiscoPhoneDevSpecific represents a different value that is assigned to the parameter m_MsgType. If you are using C instead of C++, this represents the first parameter in the structure.

Enumeration

The CiscoPhoneDevSpecificType enumeration includes valid message identifiers.

```
enum CiscoPhoneDevSpecificType
{
```

```
    CPDST_DEVICE_DATA_PASSTHROUGH_REQUEST = 1,
    CPDST_SET_DEVICE_STATUS_MESSAGES,
    CPDST_SET_DEVICE_UNICODE_DISPLAY,
    CPDST_ACQUIRE,
    CPDST_DE_ACQUIRE,
    CPDST_REQUEST_DEVICE_SNAPSHOT_INFO
};
```

Device Data PassThrough

XSI-enabled IP phones allow applications to directly communicate with the phone and access XSI features (for example, manipulate display, get user input, play tone, and so on). To allow TAPI applications to have access to some of these XSI capabilities without having to setup and maintain an independent connection directly to the phone, TAPI will provide the ability to send device data through the CTI interface. This feature gets exposed as a Cisco Unified TSP device-specific extension.

PhoneDevSpecificDataPassthrough request only gets supported for the IP phone devices. Application must open a TAPI phone device with minimum extension version 0x00030000 to make use of this feature.

The CCiscoPhoneDevSpecificDataPassThrough class is used to send the device-specific data to CTI-controlled IP phone devices.

Note Be aware that this extension requires applications to negotiate extension version as 0x00030000.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificDataPassThrough
```

Class Detail

```
class CCiscoPhoneDevSpecificDataPassThrough :
 public CCiscoPhoneDevSpecific
{
public:
    CCiscoPhoneDevSpecificDataPassThrough () :
    CCiscoPhoneDevSpecific(CPDST_DEVICE_DATA_PASSTHROUGH_REQUEST)
   {
     memset((char*)m_DeviceData, 0, sizeof(m_DeviceData)) ;
   }
   virtual ~CCiscoPhoneDevSpecificDataPassThrough() {;}
   // data size determined by MAX_DEVICE_DATA_PASSTHROUGH_SIZE
   TCHAR m_DeviceData[MAX_DEVICE_DATA_PASSTHROUGH_SIZE] ;
   // subtract out the virtual funciton table pointer size
   virtual DWORD dwSize (void) const {return (sizeof (*this)-4) ;}
}
```

Parameters

DWORD m_MsgType

Equals CPDST_DEVICE_DATA_PASSTHROUGH_REQUEST.

DWORD m_DeviceData

This character buffer contains the XML data that is to be sent to phone device.

Note Be aware that MAX_DEVICE_DATA_PASSTHROUGH_SIZE = 2000.

A phone can pass data to an application and it can get retrieved by using PhoneGetStatus (PHONESTATUS:devSpecificData). See PHONESTATUS description for further details.

Set Status Msgs

PhoneDevSpecificSetStatusMsgs allows the application to set status bit map to enable specific DEVICE_DEVSPECIFIC messages to be sent to the application.

The application must open a TAPI phone device with minimum extension version 0x00030000 to use this feature.

Note Be aware that this extension requires applications to negotiate extension version as 0x00030000.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificSetStatusMsgs
```

Class Detail

```
class CCiscoPhoneDevSpecificSetStatusMsgs:public CCiscoPhoneDevSpecific{
   public:
    CCiscoPhoneDevSpecificSetStatusMsgs() :
    CCiscoPhoneDevSpecific (CPDST_SET_DEVICE_STATUS_MESSAGES) {;}
    virtual ~CCiscoPhoneDevSpecificSetStatusMsgs() {;}
    DWORD m_DevSpecificStatusMsgFlags ; // PHONE_DEVSPECIFIC
  // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) -4) ; }
} ;
```

Parameters

DWORD m_MsgType

equals CPDST_SET_DEVICE_STATUS_MESSAGES.

DWORD m_DevSpecificStatusMsgFlags

Bit map of PHONE_DEVSPECIFIC event flag

const DWORD DEVSPECIFIC_DEVICE_DATA_PASSTHROUGH_EVENT = 0x00000001;

const DWORD DEVSPECIFIC_DEVICE_SOFTKEY_PRESSED_EVENT = 0x00000002;

const DWORD DEVSPECIFIC_DEVICE_STATE_EVENT = 0x00000004;

const DWORD DEVSPECIFIC_DEVICE_PROPERTY_CHANGED_EVENT = 0x00000008;

Set Unicode Display

PhoneDevSpecificSetUnicodeDisplay sets the Unicode display on the phone.

The application must open a TAPI phone device with minimum extension version 0x00060000 to use this feature.

Note Be aware that this extension requires applications to negotiate extension version as 0x00060000.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificSetUnicodeDisplay
```

Class Detail

```
{
public:
    CCiscoPhoneDevSpecificSetUnicodeDisplay() :
    CCiscoPhoneDevSpecific (CPDST_SET_DEVICE_UNICODE_DISPLAY) {;}
       virtual ~CCiscoPhoneDevSpecificSetUnicodeDisplay() {;}
    DWORD dwRow;
          DWORD dwColumn;
          DWORD dwSizeOfUnicodeStr;
          wchar_t UnicodeDisplay[MAX_UNICODE_DISPLAY_STRING];
  // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) -4) ; }
} ;
```

Parameters

DWORD m_MsgType

Equals CPDST_SET_DEVICE_UNICODE_DISPLAY.

DWORD m_dwRow

Row number on the phone display where the Unicode string must be displayed

DWORD m_dwColumn

Column number on the phone display where the Unicode string must be displayed

DWORD m_dwSizeOfUnicodeStr

Size of the Unicode string

wchar_t UnicodeDisplay[MAX_UNICODE_DISPLAY_STRING];

Unicode display string, with maximum size of MAX_UNICODE_DISPLAY_STRING

MAX_UNICODE_DISPLAY_STRING = 100

Explicit Acquire

The CCiscoPhoneDevSpecificAcquire class gets used to explicitly acquire any CTI controllable device.

If a Super-provider application needs to open any CTI-controllable device on the Cisco Unified Communications Manager system, the application should explicitly acquire that device by using the preceding interface. After successful response, it can open the device as usual.

Note Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificAcquire
```

Class Details

```
class CCiscoPhoneDevSpecific Acquire : public CCiscoPhoneDevSpecific{
   public:
      CCiscoPhoneDevSpecificAcquire () : CCiscoPhoneDevSpecific (CPDST_ACQUIRE)
{}
        virtual ~ CCiscoPhoneDevSpecificAcquire () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals CPDST_ACQUIRE

char m_DeviceName[16]

The DeviceName that needs to be explicitly acquired.

Explicit Deacquire

The CCiscoPhoneDevSpecificDeacquire class gets used to explicitly de-acquire an explicitly acquired device.

If a SuperProvider application explicitly acquired any CTI-controllable device on the Unified Communications Manager system, the application should explicitly de-acquire that device by using this interface.

Note Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificDeacquire
```

Class Details

```
class CCiscoPhoneDevSpecificDeacquire : public CCiscoPhoneDevSpecific{
   public:
      CCiscoPhoneDevSpecificDeacquire () : CCiscoPhoneDevSpecific (CPDST_ACQUIRE)
 {}
        virtual ~ CCiscoPhoneDevSpecificDeacquire () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals CPDST_DEACQUIRE

char m_DeviceName[16]

The DeviceName that needs to be explicitly de-acquired.

Request Call RTP Snapshot

The CCiscoPhoneDevSpecificGetRTPSnapshot class gets used to request Call RTP snapshot event from the device. There will be LineCallDevSpecific event for each call on the device.

Note Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+--CCiscoPhoneDevSpecificGetRTPSnapshot
```

Class Details

```
class CCiscoPhoneDevSpecificGetRTPSnapshot:
 public CCiscoPhoneDevSpecific{
    public:
   CCiscoPhoneDevSpecificGetRTPSnapshot () :
    CCiscoPhoneDevSpecific (CPDST_REQUEST_RTP_SNAPSHOT_INFO) {}
        virtual ~ CCiscoPhoneDevSpecificGetRTPSnapshot () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

Parameters

DWORD m_MsgType

Equals CPDST_DEACQUIRE

char m_DeviceName[16]

The DeviceName that needs to be explicitly de-acquired.

Messages

This section describes the line device specific messages that the Cisco Unified TSP supports. An application receives nonstandard TAPI messages in the following LINE_DEVSPECIFIC messages:

- A message to signal when to stop and start streaming RTP audio.
- A message that contains the call handle of active calls when the application starts up.
- A message that indicates to set the RTP parameters based on the data of the message.
- A message that indicates secure media status.

The message type represents an enumerated integer with the following values:

```
enum CiscoLineDevSpecificMsgType{
SLDSMT_START_TRANSMISION = 1,
SLDSMT_STOP_TRANSMISION,
SLDSMT_START_RECEPTION,
SLDSMT_STOP_RECEPTION,
SLDST_LINE_EXISTING_CALL,
SLDSMT_OPEN_LOGICAL_CHANNEL,
SLDSMT_CALL_TONE_CHANGED,
SLDSMT_LINECALLINFO_ DEVSPECIFICDATA,
SLDSMT_HOLD_REVERSION,
SLDSMT_LINE_PROPERTY_CHANGED,
SLDSMT_MONITORING_STARTED,
SLDSMT_MONITORING_ENDED,
SLDSMT_RECORDING_STARTED,
SLDSMT_RECORDING_ENDED,
SLDSMT_NUM_TYPE,
SLDSMT_IP_ADDRESSING_MODE_CHANGED,
SLDSMT_START_TRANSMISION_ADDRESSING_MODE,
SLDSMT_START_RECEPTION_ADDRESSING_MODE,
SLDSMT_DEVICE_STATE,
SLDSMT_MONITORING_TERMINATED,
SLDSMT_MEDIA_TO_BIB_STARTED,
SLDSMT_MEDIA_TO_BIB_ENDED,
SLDSMT_MONITORING_MODE_UPDATED,
SLDSMT_RTP_GET_IP_PORT
SLDSMT_MULTIMEDIA_STREAMSDATA,
SLDSMT_ANNOUNCEMENT_STARTED,
SLDSMT_ANNOUNCEMENT_ENDED,
SLDSMT_RECORDING_FAILED
};
```

Announcement Events

SLDSMT_ANNOUNCEMENT_STARTED

When an announcement starts, the SLDSMT_ANNOUNCEMENT_STARTED message is sent to the application. The format of the parameters follows:

LINE_DEVSPECIFIC

hDevice -TAPI call handle

dwParam1 - **SLDSMT_ANNOUNCEMENT_STARTED**

dwParam2 -unused

dwParam3 -unused

SLDSMT_ANNOUNCEMENT_ENDED

When an announcement ends, the SLDSMT_ANNOUNCEMENT_ENDED message is sent to the application. If the announcement does not play, the application analyzes the cause code parameter to verify whether the announcement was successful and the failure reason. The format of the parameters follows:

LINE_DEVSPECIFIC

hDevice -TAPI call handle

dwParam1 - **SLDSMT_ANNOUNCEMENT_ENDED**

dwParam2 -result (0 or error code in case of failure)

dwParam3 -unused

SLDSMT_RECORDING_FAILED

An SLDSMT_RECORDING_FAILED is generated to the application when a recording is unable to restart after being interrupted. The application must then restart the recording. The format of the parameters follows:

LINE_DEVSPECIFIC

hDevice -TAPI call handle

dwParam1 -SLDSMT_RECORDING_FAILED (0x1C)

dwParam2 -cause code for failure

dwParam3 -unused

Start Transmission Events

SLDSMT_START_TRANSMISION

When a message is received, the RTP stream transmission starts and:

- dwParam2 specifies the network byte order IP address of the remote machine to which the RTP stream should be directed.
- dwParam3, specifies the high-order word that is the network byte order IP port of the remote machine to which the RTP stream should be directed.
- dwParam3, specifies the low-order word that is the packet size, in milliseconds, to use.

The application receives these messages to signal when to start streaming RTP audio. At extension version 1.0 (0x00010000), the parameters have the following format:

- dwParam1 contains the message type.

- dwParam2 contains the IP address of the remote machine.
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 2.0 (0x00020000), start transmission uses the following format:

- dwParam1:from highest order bit to lowest
- First two bits blank
- Precedence value 3 bits
- Maximum frames per packet 8 bits
- G723 bit rate 2 bits
- Silence suppression value 1 bit
- Compression type 8 bits
- Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 4.0 (0x00040000), start transmission has the following format:

- hCall – The call of the Start Transmission event
- dwParam1:from highest order bit to lowest
  - First two bits blank
  - Precedence value 3 bits
  - Maximum frames per packet 8 bits
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

Start Reception Events

SLDSMT_START_RECEPTION

When a message is received, the RTP stream reception starts and:

- dwParam2 specifies the network byte order IP address of the local machine to use.
- dwParam3, specifies the high-order word that is the network byte order IP port to use.
- dwParam3, specifies the low-order high-order word that is the packet size, in milliseconds, to use.

When a message is received, the RTP stream reception should commence.

At extension version 1, the parameters have the following format:

- dwParam1 contains the message type.
- dwParam2 contains the IP address of the remote machine.
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 2 start reception uses the following format:

- dwParam1:from highest order bit to lowest
- First 13 bits blank
- G723 bit rate 2 bits
- Silence suppression value 1 bit
- Compression type 8 bits
- Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 4.0 (0x00040000), start reception uses the following format:

- hCall – The call of the Start Reception event
- dwParam1:from highest order bit to lowest
  - First 13 bits blank
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

Stop Transmission Events

SLDSMT_STOP_TRANSMISION

When a message is received, transmission of the streaming should stop.

At extension version 1.0 (0x00010000), stop transmission uses the following format:

- dwParam1 – Message type

At extension version 4.0 (0x00040000), stop transmission uses the following format:

- hCall – The call for which the Stop Transmission event applies.
- dwParam1 – Message type

Stop Reception Events

SLDSMT_STOP_RECEPTION

When a message is received, reception of the streaming should stop.

At extension version 1.0 (0x00010000), stop reception uses the following format:

- dwParam1 -message type

At extension version 4.0 (0x00040000), stop reception uses the following format:

- hCall – The call for which the Stop Reception event applies.
- dwParam1 – Message type

Existing Call Events

SLDST_LINE_EXISTING_CALL

These events inform the application of existing calls in the PBX when it starts up. The format of the parameters follows:

- dwParam1 – Message type
- dwParam2 – Call object
- dwParam3 – TAPI call handle

Open Logical Channel Events

SLDSMT_OPEN_LOGICAL_CHANNEL

When a call has media established at a CTI Port or Route Point that is registered for Dynamic Port Registration, receipt of this message indicates that an IP address and UDP port number need to be set for the call.

Note This extension is only available if extension version 0x00040000 or higher gets negotiated.

The following format of the parameters applies:

- hCall -The call for which the Open Logical Channel event applies
- dwParam1 – Message type
- dwParam2 – Compression Type
- dwParam3 – Packet size in milliseconds

At extension version 9.0 (0x00090000), start transmission has the following format:

- hCall -The call the Open Logical Channel event is for
- dwParam1: from highest order bit to lowest
- First eight bits blank
- Maximum frames per packet 8 bits
- Compression type 8 bits
- Message type 8 bits
- dwParam2 contains the IP addressing mode
- dwParam3: Packet size in milliseconds

At extension version B.0 (0x000B0000), Open Logical channel has the following format:

- hCall -The call the Open Logical Channel event is for
- dwParam1: from highest order bit to lowest
- First sixteen bits blank
- Compression type 8 bits
- Message type 8 bits
- dwParam2: from highest order bit to lowest
- First twenty three bits blank
- SetRTPInfo (twenty fourth bit from MSB/ninth bit from LSB)
- IP addressing mode 8 bits
- dwParam3: Packet size in milliseconds

LINECALLINFO_DEVSPECIFICDATA Events

SLDSMT_LINECALLINFO_ DEVSPECIFICDATA

This message indicates that the DEVSPECIFICDATA information is changed in the DEVSPECIFIC portion of the LINECALLINFO structure for the different fields.

Note The fields are only available if the negotiated version contains support for the particular feature.

The following format applies for the parameters:

- hCall -The call handle
- dwParam1 -Message type
- dwParam2

```
SLDST_SRTP_INFO | SLDST_QOS_INFO |SLDST_PARTITION_INFO |
SLDST_EXTENDED_CALL_INFO | SLDST_CALL_ATTRIBUTE_INFO|SLDST_CCM_CALLID|
SLDST_CALL_SECURITY_STATUS | SLDST_NUMBER_TYPE_CHANGED |
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED |
SLDST_FAR_END_IP_ADDRESS_CHANGED | SLDST_UNIQUE_CALL_REF_ID_INFO
SLDST_DEVICE_VIDEO_CAP_INFO | SLDST_MULTIMEDIA_STREAMS_INFO
```

The bit mask values follow:

SLDST_SRTP_INFO	0x00000001
SLDST_QOS_INFO	0x00000002
SLDST_PARTITION_INFO	0x00000004
SLDST_EXTENDED_CALL_INFO	0x00000008
SLDST_CALL_ATTRIBUTE_INFO	0x00000010
SLDST_CCM_CALL_ID	0x00000020
SLDST_SECURITY_STATUS_INFO	0x00000040
SLDST_NUMBER_TYPE_CHANGED	0x00000080
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED	0x00000100
SLDST_FAR_END_IP_ADDRESS_CHANGED	0x00000200
SLDST_UNIQUE_CALL_REF_ID_INFO	0x00000400
SLDST_DEVICE_VIDEO_CAP_INFO	0x00000800
SLDST_MULTIMEDIA_STREAMS_INFO	0x00001000

For example, if there are changes in SRTP and QoS but not in Partition, then both the SLDST_SRTP_INFO and SLDST_QOS_INFO bits are set. The value for dwParam2 = SLDST_SRTP_INFO | SLDST_QOS_INFO = 0x00000011

- dwParam3 -If a change occurs in the SRTP information, then this field contains the CiscoSecurityIndicator.

```
enum CiscoSecurityIndicator
{
SRTP_MEDIA_ENCRYPT_KEYS_AVAILABLE,
SRTP_MEDIA_ENCRYPT_USER_NOT_AUTH,
SRTP_MEDIA_ENCRYPT_KEYS_UNAVAILABLE,
SRTP_MEDIA_NOT_ENCRYPTED
};
```

Note dwParam3 is used when dwParam2 has the SRTP bit mask set.

Call Tone Changed Events

SLDSMT_CALL_TONE_CHANGED

When a tone change occurs on a call, receipt of this message indicates the tone and the feature that caused the tone change.

Note Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated. In the Cisco Unified TSP 4.1 release and later, this event only gets sent for Call Tone Changed Events where the tone equals CTONE_ZIPZIP and the tone gets generated as a result of the FAC/CMC feature.

The format of the parameters follows:

- hCall—The call for which the Call Tone Changed event applies
- dwParam—Message type
- dwParam2—CTONE_ZIPZIP, 0x31 (Zip Zip tone), CTONE_ZIP, 0x32 (Zip tone)
- dwParam3—If dwParam2 is CTONE_ZIPZIP, this parameter contains a bitmask with the following possible values:
  - CZIPZIP_FACREQUIRED—If this bit is set, it indicates that a FAC is required.
  - CZIPZIP_CMCREQUIRED—If this bit is set, it indicates that a CMC is required.
  - If dwParam2 is CTONE_ZIP, this parameter contains direction mode with the following possible values:

    0 -Tone is played at local End

    1 -Tone is played at Remote End

Note For a DN that requires both codes, the first event always applies for the FAC and CMC code. The application optionally can send both codes separated by # in the same request. The second event generation remains optional based on what the application sends in the first request.

Line Property Changed Events

SLDSMT_LINE_PROPERTY_CHANGED

When a line property is changed, a LINEDEVSPECIFIC event is fired with indication of the changes.

Note This extension is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

dwParam1 -Message type

dwParam2 -indication type -CiscoLinePropertyChangeType

```
enum CiscoLinePropertyChangeType
{
LPCT_INTERCOM_LINE                  =   0x00000001,
LPCT_RECORDING_TYPE                 =   0x00000002,
LPCT_MAX_CALLS                      =   0x00000004,
LPCT_BUSY_TRIGGER                   =   0x00000008,
LPCT_LINE_INSTANCE                  =   0x00000010,
LPCT_LINE_LABEL                     =   0x00000020,
LPCT_VOICEMAIL_PILOT                =   0x00000040,
LPCT_DEVICE_IPADDRESS               =   0x00000080,
LPCT_NEWCALL_ROLLOVER               =   0x00000100,
LPCT_CONSULTCALL_ROLLOVER           =   0x00000200,
LPCT_JOIN_ON_SAME_LINE              =   0x00000400,
LPCT_JOIN_ACROSS_LINE               =   0x00000800,
LPCT_DIRECTTRANSFER_ON_SAME_LINE    =   0x00001000,
LPCT_DIRECTTRANSFER_ACROSS_LINE     =   0x00002000
};
```

dwParam3 -default = 0,

In case, dwParam2 = LPCT_INTERCOM_LINE, dwParam3 is the result of the change

```
Enum CiscoIntercomLineChangeResult
{
IntercomSettingChange_successful = 0;
IntercomSettingRestorationFail = 1
}
```

If dwParam2 = LPCT_RECORDING_TYPE, dwParam3 will have a new Recording Type:

```
enum recordType
{
RecordType_NoRecording =                        0,
```

```
RecordType_AutomaticRecording =                 1,
RecordType_ApplicationInvokedCallRecording =    2,
RecordType_DeviceInvokedCallRecording =         3
};
```

Phone Property Changed Events

CPDSMT_PHONE_PROPERTY_CHANGED_EVENT

When a Phone property is changed, a PHONE_DEVSPECIFIC event is fired with indication of what has been changed.

The following format of the parameters applies:

dwParam1 – SLDSMT_LINE_PROPERTY_CHANGED (0x04)

dwParam2 – indication type – CiscoLinePropertyChangeType

dwParam3 – updated based on dwParam2 (LinePropertyChangeType)

CiscoPhonePropertyChangeType

```
enum CiscoPhonePropertyChangeType
{
    PPCT_DEVICE_IPADDRESS =             0x00000001,
    PPCT_NEWCALL_ROLLOVER =             0x00000002,
    PPCT_CONSULTCALL_ROLLOVER =         0x00000004,
    PPCT_JOIN_ON_SAME_LINE =            0x00000008,
    PPCT_JOIN_ACROSS_LINE =             0x00000010,
    PPCT_DIRECTTRANSFER_ON_SAME_LINE =  0x00000020,
    PPCT_DIRECTTRANSFER_ACROSS_LINE =   0x00000040,
    PPCT_DEVICE_MULTIMEDIACAP_INFO =    0x00000080,

};
```

Monitoring Started Event

SLDSMT_MONITORING_STARTED

When monitoring starts on a particular call, this event is triggered for the monitored call to inform the application.

Note This event is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—0
- dwParam3—0

Monitoring Ended Event

SLDSMT_MONITORING_ENDED

When monitoring is stopped for a particular call, this event is triggered for the monitored call to inform the application.

Note This event is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—Reason code
- dwParam3—0

Recording Started Event

SLDSMT_RECORDING _STARTED

When recording starts on a particular call, this event is triggered to inform the same to the application.

Note This event is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—0
- dwParam3—0

Recording Ended Event

SLDSMT_RECORDING _ENDED

When recording is stopped on a particular call, this event is triggered to inform the same to the application.

Note This event is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—Reason code

- dwParam3—0

Recording Failure Event

SLDSMT_RECORDING_FAILED

When a recording is started and another feature can cause the recording to stop and start again. If the recording does not restart, an SLDSMT_RECORDING_FAILED message is generated to the application. The application then restarts the recording.

The format of the parameters follows:

- **LINE_DEVSPECIFIC**
- hDevice -TAPI call handle
- dwParam1 - SLDSMT_RECORDING_FAILED (0x1C)
- dwParam2 -cause code for failure
- dwParam3 -unused

Silent Monitoring Session Terminated Event

SLDSMT_MONITORING_TERMINATED

When Monitoring Session is toned down as security capabilities of the supervisor do not meet or exceed the capabilities of agent, this event is fired on the supervisor to inform the same to the application.

Note This event is only available if extension version 0x000A0000 or higher is negotiated.

The format of the parameters follows:

- dwParam1 – Message type -SLDSMT_MONITORING_TERMINATED
- dwParam2 – TransactionID – which is unique for the Monitoring session
- dwParam3 – New Cause Code -LINEDISCONNECTMODE_INCOMPATIBLE

Media to BIB Started Event

SLDSMT_MEDIA_TO_BIB_STARTED

This event indicates that agent greeting call has been successfully set up.

Note This event is only available if extension version 0x000B0000 or higher is negotiated.

The format of the parameters follows:

- dwParam1 – Message type -SLDSMT_MEDIA_TO_BIB_STARTED

- dwParam2 – reserved (0)
- dwParam3 – reserved (0)

Media to BIB Ended Event

SLDSMT_MEDIA_TO_BIB_ENDED

This event indicates that the agent greeting has ended.

Note This event is only available if extension version 0x000B0000 or higher is negotiated.

The format of the parameters follows:

- dwParam1 – Message type -SLDSMT_MEDIA_TO_BIB_ENDED
- dwParam2 – result code:
  - non 0: Agent Greeting was successfully played
  - 0: Agent Greeting was not successfully played dwParam3 – result code

Get IP and Port Event

SLDSMT_RTP_GET_IP_PORT

This event indicates that the application has to set the RTP Port and IP information using existing SetRTP devspecific Extension. The application has to set the RTP information only for Dynamically Registered CTI Ports or Route Points and for static Registered CTI Ports, application has to open the port used for registration.

Note This event is available only if extension version 0x000B0000 or higher is negotiated.

The format of the parameters follows:

- dwParam1 – Message type -SLDSMT_RTP_GET_IP_PORT
- dwParam2 – IP Addressing Capability (from highest order bit to lowest)
- First twenty three bits blank
- SetRTPInfo (twenty fourth bit from MSB or ninth bit from LSB)
- IP addressing mode 8 bits
- dwParam3 – reserved (0)

MultiMedia Streams Data Notification Event

SLDSMT_MULTIMEDIA_STREAMSDATA

When MultiMediaStreams Data Information is changed on a Call, SLDSMT_MULTIMEDIA_STREAMSDATA message is sent to the application.

The format of the parameters follows:

LINE_DEVSPECIFIC

hDevice – TAPI call handle

dwParam1 – **SLDSMT_MULTIMEDIA_STREAMSDATA**

dwParam2 – unused

dwParam3 – unused

Monitor Mode Update Event

SLDSMT_MONITORING_MODE_UPDATED

This event indicates that the monitoring mode has been successfully updated to the value in dwParam1 and is sent to active supervisor and agent lines.

Note This event is available only if extension version 0x000B0000 or higher is negotiated.

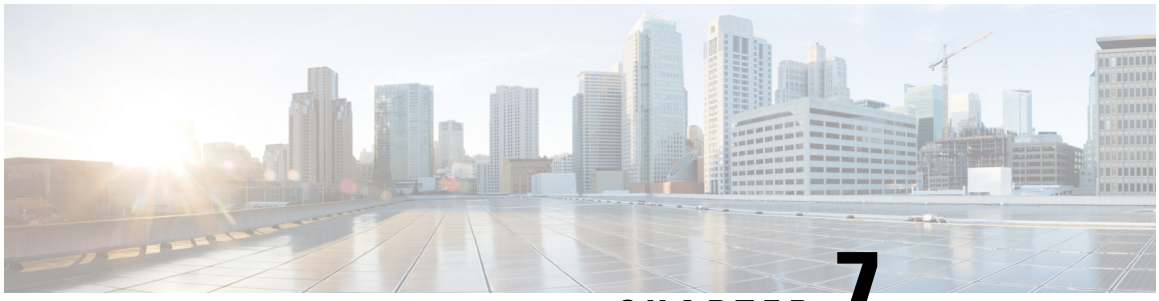
The format of the parameters follows:

- dwParam1 – Message type -SLDSMT_MONITORING_MODE_UPDATED
- dwParam2 – monitoring mode

```
    enum
    {
        MonitorMode_Silent  = 1,
        MonitorMode_Whisper = 2,
        MonitorMode_Active  = 3    //  Not currently used
    } MonitorMode;
```

- dwParam3 – active tone direction

```
enum
    {
         PlayToneDirection_NoLocalOrRemote = 0,
         PlayToneDirection_LocalOnly,
         PlayToneDirection_RemoteOnly,
         PlayToneDirection_BothLocalAndRemote
    } PlayToneDirection
```



CHAPTER 7

Cisco TSP Media Driver

Cisco Media Driver introduces a new and innovative way for TAPI-based applications to provide media interaction such as play announcements, record calls, and so on.

Cisco TSP 8.0(1) includes support for both Cisco Media Driver and Cisco Wave Driver, but only one driver can be active at any given time.

Cisco Media Driver offers several advantages:

- Simplified Installation and Management—Cisco Media Driver configuration can be completed through the Cisco TSP Installation Wizard. Channel and port settings are consistently and automatically applied to all configured TSP instances.
- Performance and Scalability—Cisco Media Driver can scale to support up to 1000 configured ports with hundreds of simultaneously active media channels. Refer to the application vendor's Installation Guide to determine the number of channels supported by the TAPI application.
- Codec Support—Cisco Media Driver supports 8KHz, 16-bit PCM, G.711 a-law, G.711 u-law natively. Additionally, G.729a can be supported when pass-through mode is enabled.
- Reliability—Cisco Media Driver runs as an independent process, similar to Windows applications, providing greater application stability and reliability. Creating and debugging media applications is now much easier.

Cisco Rtp Library Components

Header Files

The following header files contain declaration of all functions, data structures, etc. exposed by Cisco Rtp Library.

- ciscortpapi.h
- ciscortpbase.h
- ciscortpcbcs.h
- ciscortpcodec.h
- ciscortperr.h
- ciscortpep.h
- ciscortpip.h

In order to use Cisco Rtp Library functionality a typical application would only need to explicitly include ciscortpapi.h and ciscortpep.h files.

Import Library

The following import library has to be linked with an application in order to use Cisco Rtp Library functionality:

- cmrtplib.lib

DLLs

The following DLLs are installed as a part of CiscoTSP plug-in and used by Cisco Rtp Library:

Windows 32bit OS (x86):

- ciscortplib.dll
- ciscortpmon.dll
- ciscortpg711a.dll
- ciscortpg711u.dll
- ciscortpg729.dll
- ciscortppcm16.dll

Windows 64bit OS (x64):

- ciscortplib64.dll
- ciscortpmon64.dll
- ciscortpg711a64.dll
- ciscortpg711u64.dll
- ciscortpg72964.dll
- ciscortppcm1664.dll

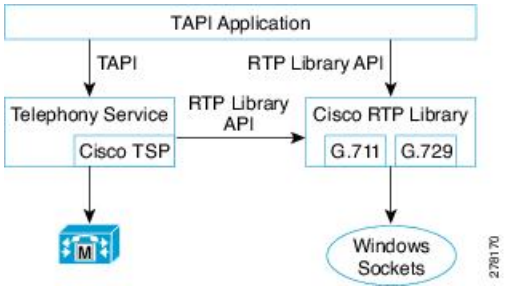
TAPI Application Support

CiscoTSP and Cisco Rtp Library Interaction

In order to allow TAPI applications to associate TAPI line device with Rtp Library media endpoints Cisco TSP implements two new device classes: ciscowave/in and ciscowave/out. If TAPI line device is capable to terminate media by means of Cisco Rtp Library, an application can use ciscowave/in and ciscowave/out device class names in the TAPI lineGetID() function to obtain associated media device identifiers. Media device identifier can be used in Cisco Rtp Library APIs to create media endpoints and manipulate media on a corresponding TAPI line device.

The following figure shows high level view of TAPI application which uses Cisco TAPI service provider and Cisco Rtp Library functionalities.

Figure 30: TAPI Application with Cisco Components



Codec Advertisement

Cisco Media Driver devices advertise G.711 support natively. Cisco Unified CM automatically invokes Media Termination Points (MTPs) when needed to provide transcoding (see Example 1). If MTPs are not configured and transcoding is required, call setup fails (see Example 2).

Example 1

1. G729PassThrough set to OFF (default).
2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 support (default).
4. Unified CM is configured with MTPs, which can be used if transcoding is needed.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729, so an MTP is inserted to provide transcoding.

Example 2

1. G729PassThrough set to OFF (default).

2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 support (default).
4. Unified CM is not configured with MTPs for transcoding.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729 and no MTPs are available, so call setup fails.

Applications which natively support G.729 can change the default codec advertisement by setting the G729PassThrough registry option to ON (1).

The TSP application is then responsible for playing the appropriate media file (G.711 or G.729) based on the compatible codecs supported by the Device receiving the media (see Example 3 below).

The Registry key can be found at:

- Windows XP: HKEY_Local_Machine/Software/Cisco Systems, Inc./ RtpLib/G729PassThrough
- Windows Vista: HKEY_USERS\S-1-5-20\Software\Cisco Systems, Inc.\ RtpLib\G729PassThrough

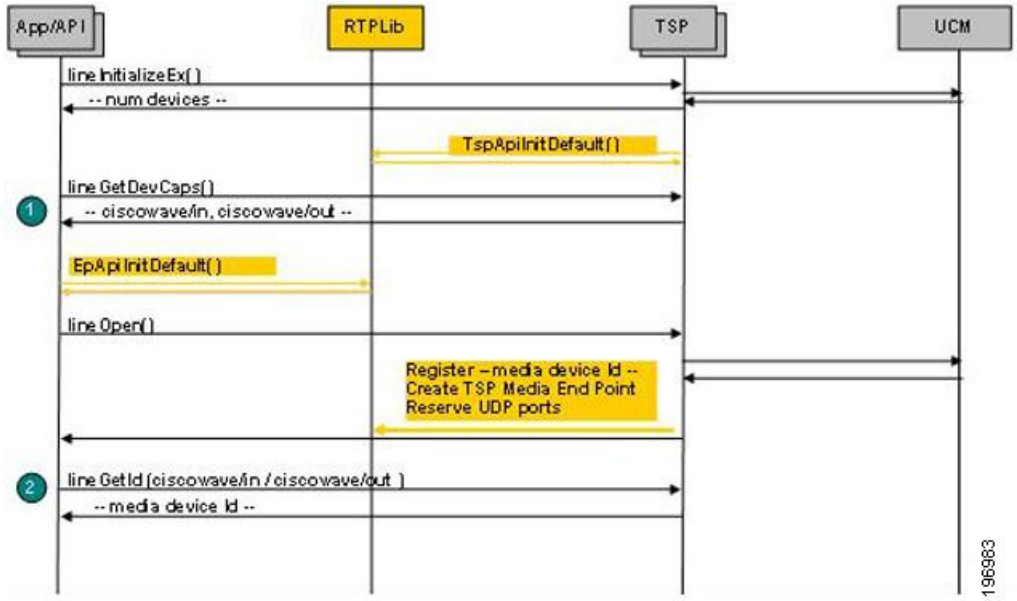
Example 3

1. G729PassThrough set to ON.
2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 and G.729 support.
4. Unified CM is not configured with MTPs for transcoding.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729, so the application plays the appropriate G.729 media file.

Typical TAPI Application Message Flow

The message flow in the following figure is described in steps 1 and 2.

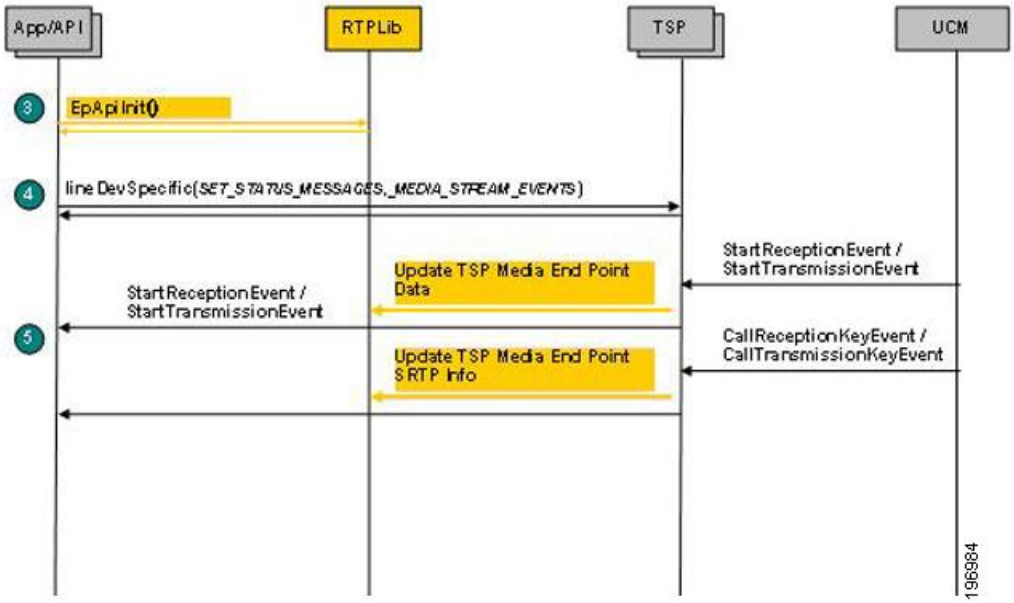
Figure 31: Typical TAPI Application Message Flow 1



1. Initialize TAPI, get LINEINFO for available line devices, find devices which are capable of using Cisco Rtp Library functionalities
2. Get media device identifier associated with a particular line device

   The message flow in the following figure is described in steps 3 to 5.

Figure 32: Typical TAPI Application Message Flow 2

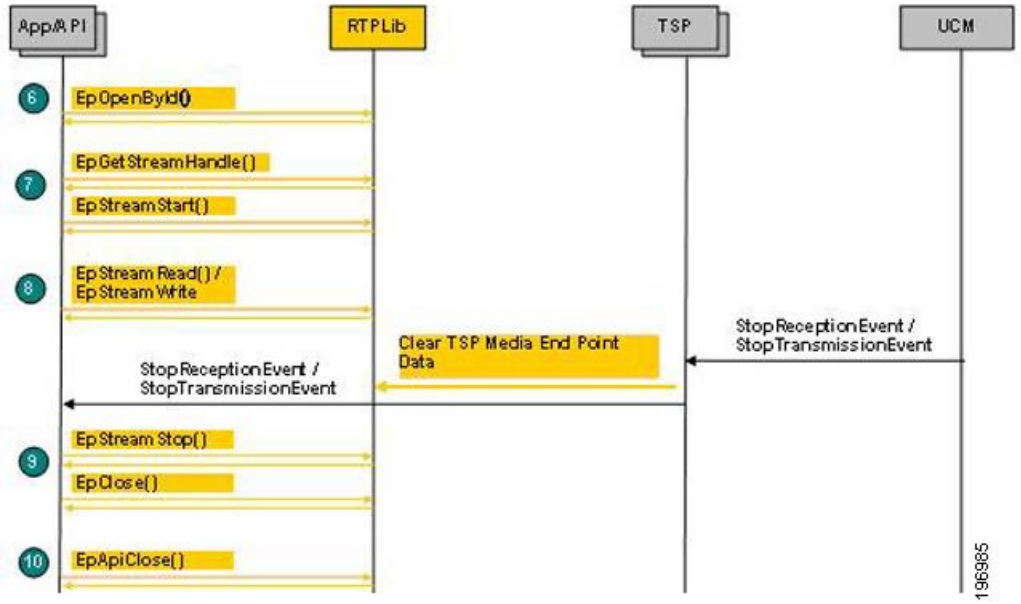


3. Initialize Rtp Library
4. Subscribe for media stream events for relevant devices using Cisco lineDevSpecific extension

5. Start monitoring media events

   The message flow in following figure is described in steps 6 to 10.

   Figure 33: Typical TAPI Application Message Flow 3



6. Create media endpoint
7. Get in/out stream handle and start data streaming
8. Receive / transmit data
9. Stop data streaming, close endpoint
10. Close EpAPI before exiting the program

EpAPI Functions

EpApiInit

Initializes EpApi and Rtp Library.

Syntax

```
CMAPI bool     EpApiInit (
PRTPLIBTRACE   pTraceCallback,
USHORT         portRangeStart,
USHORT         numPorts,
int            IPAddressFamily,
PRTPADDR       pDefaultRtpAddr
);
```

Parameters

pTraceCallback

Pointer to an application callback function to be called with the trace record data passed to it.

portRangeStart

First port number in the continuous range of UDP ports (port pool) which can be used to create endpoints.

numPorts

Number of ports in the UDP port range (port pool) which can be used to create endpoints.

IPAddressFamily

IP address family to be used by Rtp Library to create endpoints can be set to:

- AF_UNSPEC: both AF_INET and AF_INET6 can be used.
- AF_INET: AF_INET only can be used.
- AF_INET6: AF_INET6 only can be used.

This settings can be overwritten by the pDefaultRtpAddr parameter.

pDefaultRtpAddr

IP address to be used use by Rtp Library to create endpoints. If not NULL, only this address will be used.

Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_ADDR_NOTAVAIL	Unable to create endpoint with specified IP address family
EP_ERR_PARAM_INVALID	The following describes a possible cause of the error: Invalid number of UDP ports
RTP_ERR_INITALREADY	Already initialized
RTP_ERR_TIMER_NOTAVAIL	Unable to create high resolution timer

Remarks

An error code can be set even when EpApiInit returns true. In some cases a default action / value can be assumed even if a parameter or registry settings is invalid. In those cases EpApiInit returns true but also set a proper error code to indicate an issue.

EpApiInitByDefault

Initializes EpApi and Rtp Library with default settings.

Syntax

```
CMAPI bool EpApiInitByDefault (
    PRTPLIBTRACEpTraceCallback,
);
```

Parameters

pTraceCallback

Pointer to an application callback function to be called with the trace record data passed to it.

Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
RTP_ERR_INITALREADY	Already initialized

Remarks

Rtp Library will be initialized as if registry is set as follows:

- UDPPortRangeStart = 50000
- UDPPortRangeEnd = 50999

EpApiClose

Closes EpApi.

Syntax

```
CMAPI bool EpApiClose ();
```

Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.

Remarks

As a result of this function execution all active sessions, connections and streams will be terminated, timers closed and all data freed.

EpLocalAddressGetAll

Returns an array of RTPADDR structures which contain local IP addresses available for use by Rtp Library.

Syntax

```
CMAPI int EpLocalAddressPortGetAll(
  PRTPADDR pBuffer,
  int *    pBufSize
);
```

Parameters

pBuffer

Pointer to a memory buffer to fill in with array of RTPADDR structures or NULL.

pBufSize

- IN—Length of the buffer (in bytes), pointed to by pBuffer.
- OUT—Space in the buffer used or required.

Return Value

If no errors occurs, this function returns a number of available local IP addresses and an array of RTPADDR structures in the pBuffer.

If pBuffer parameter value is NULL, the function returns the number of available local IP addresses and the pBufSize will contain the buffer size required for the RTPADDR structure array.

If an error occurs, 0 is returned and a specific error code can be retrieved by calling EpApiGetLastError. In case of EP_ERR_PARAM_INVALID error, pBufSize will contain the size of the required buffer.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	Buffer is not large enough.

EpLocalAddressPortGet

Reserves port from the port pool and returns it together with a local IP address.

Syntax

```
CMAPI PRTPADDR EpLocalAddressPortGet();
```

Return Value

If no errors occurs, this function returns pointer to RTPADDR structure with the first (or default) local IP address used by Rtp Library and reserved UDP port number.

If an error occurs, NULL is returned and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

EpLocalAddressPortGetByFamily

Reserves port from the port pool and returns it alone with a local IP address for the specified family.

Syntax

```
CMAPI PRTPADDR EpLocalAddressPortGetByFamily(
  int          IPAddressFamily
);
```

Parameters

IPAddressFamily

IP address family: AF_INET or AF_INET6

Returns: Pointer to RTPADDR structure or NULL.

Return Value

If no errors occurs, this function returns pointer to RTPADDR structure with the first local IP address for the specified family used by Rtp Library and reserved UDP port number. If an error occurs, NULL is returned and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
RTP_ERR_ADDR_NOTAVAIL	No IP address available for specified family
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

EpLocalAddressPortGetByIdx

Reserves UDP port from the Rtp Library port pool and returns it in the RTPADDR data structure alone with the IP address of the network interface card specified by the index parameter.

Syntax

```
CMAPI PRTPADDR EpLocalAddrPortGetByIdx (
  int          index
);
```

Parameters

index

Index in the list of available local network addresses returned by EpLocalAddressGetAll function call.

Return Value

If no error occurs, this function returns pointer to RTPADDR structure which contains local IP address and reserved UDP port number. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	Invalid index value.
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

Remarks

List of available local network addresses can be obtained by EpLocalAddressGetAll function call.

EpLocalAddrPortFree

Returns local UDP port previously reserved by EpLocalAddressGet, EpLocalAddressGetByIdx or EpLocalAddressGetByFamily back to the port pool.

Syntax

```
CMAPI bool EpLocalAddrPortFree (
  PRTPADDR pLocalAddrPort
);
```

Parameters

pLocalAddrPort

Pointer to RTPADDR data structure which contains port number of previously reserved local UDP port.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.

Remarks

Local UDP could is reserved by EpLocalAddressGet, EpLocalAddressGetByIdx or EpLocalAddressGetByFamily.

EpOpenById

Creates media endpoint based on TSP data associated with the specified media device identifier.

Syntax

```
CMAPI HANDLE EpOpenById (
  DWORD             deviceId,
  StreamDirection   streamDir,
  PRTPDATACALLBACK  pCallback
);
```

Parameters

deviceId

Media device identifier obtained by calling TAPI lineGetID() for ciscowave/in or ciscowave/out device class.

streamDir

Stream direction. This parameter can be one of the following values:

- ToApp
- ToNwk
- Both

pCallback

Pointer to an application callback function to be called when data buffer is received/sent or an error occurred.

Return Value

If no errors occurs, this function returns a handle which can be used to reference the endpoint. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	The following describes a possible cause of the error: • Specified device identifier is invalid. • Required data associated with device identifier data is missing • Specified stream direction is invalid

Remarks

Endpoint is created based on a data associated by TSP with the deviceId.

EpClose

Close endpoint created by EpOpen.

Syntax

```
CMAPI bool EpClose (
  HANDLE   hEp
);
```

Parameters

hEp

Endpoint handle returned by EpOpen.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.

EpGetStreamHandle

Returns endpoint stream handle for a specified stream type and direction

Syntax

```
CMAPI HANDLE EpGetStreamHandle (
  HANDLE           hEp,
  StreamType       streamType,
  StreamDirection  streamDir
);
```

Parameters

hEp

Endpoint handle returned by EpOpen.

streamType

Stream type. This parameter can be one of the following values:

- STREAM_TYPE_AUDIO
- STREAM_TYPE_VIDEO

streamDir

Stream direction. This parameter can be one of the following values:

- ToApp
- ToNwk

Return Value

If no errors occurs, this function returns stream handle which can be used to reference the stream. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.
EP_ERR_PARAM_INVALID	Specified stream type or direction is invalid.

EpStreamStart

Enables data flow on a specified stream

Syntax

```
CMAPI bool EpStreamStart (
  HANDLE            hStream,
  PRTPDATACALLBACK  pCallback
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pCallback

Pointer to an application callback function to be called when data buffer is received/sent or an error occurred.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.
EP_ERR_ADDR_INUSE	Address (protocol-IPaddress-port) is already in use.

Remarks

EpStreamStart() should be explicitly called by an application in order to stream data flow (open socket, port). It is not done implicitly by the Rtp Library as it was done before by the Cisco kernel mode wave driver.

EpStreamStop

Disables data flow on a specified stream.

Syntax

```
CMAPI bool EpStreamStop (
  HANDLE   hStream
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

Remarks

EpStreamStop() should be explicitly called by an application in order to disable stream data flow. It is not done implicitly by the Rtp Library as it was done before by the Cisco kernel mode wave driver.

EpStreamRead

Read data from a stream.

Syntax

```
CMAPI bool EpStreamRead (
  HANDLE            hStream,
  PUCHAR            pBuffer,
  int               bufLen,
  PVOID             pAppData,
  PRTPDATACALLBACK  pCallback
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pBuffer

Pointer to a buffer for incoming data.

bufLen

Buffer size.

pAppData

Pointer to an application data area. It will be associated with the buffer and will be passed back to the application callback function as the pAppData parameter.

pCallback

Pointer to an application callback function to be called when data buffer is received or an error occurred.

Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpStreamWrite

Write data to a stream.

Syntax

```
CMAPI bool EpStreamWrite (
  HANDLE           hStream,
  PUCHAR           pBuffer,
  int              bufLen,
  PVOID            pAppData,
```

```
  PRTPDATACALLBACK pCallback
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pBuffer

Pointer to a buffer which contains data.

bufLen

Data length

pAppData

Pointer to an application data area. It will be associated with the buffer and will be passed back to the application callback function as the pAppData parameter.

pCallback

Pointer to an application callback function to be called when data buffer has been written or an error occurred.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpStreamCodecInGet

Returns stream inbound codec format information.

Syntax

```
CMAPI bool EpStreamCodecInGet (
  HANDLE         hStream,
  PWAVEFORMATEX  pWaveFormat
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pWaveFormat

Pointer to a WAVEFORMATEX data structure. Upon successful completion of the request this structure is filled with stream inbound codec format data.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpStreamCodecInSet

Sets stream inbound codec format.

Syntax

```
CMAPI bool EpStreamCodecInSet (
  HANDLE         hStream,
  PWAVEFORMATEX  pWaveFormat,
  ULONG          pktSizeMs
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pWaveFormat

Pointer to a WAVEFORMATEX data structure which contains codec information.

pktSizeMs

Packet size in milliseconds. If value 0 (zero) is specified a default value (20) is used.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpStreamCodecOutGet

Returns stream outbound codec format information.

Syntax

```
CMAPI bool EpStreamCodecOutGet (
  HANDLE         hStream,
  PWAVEFORMATEX  pWaveFormat
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pWaveFormat

Pointer to a WAVEFORMATEX data structure. Upon successful completion of the request this structure is filled with stream outbound codec format data.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpStreamCodecOutSet

Sets stream outbound codec format.

Syntax

```
CMAPI bool EpStreamCodecInSet (
  HANDLE         hStream,
  PWAVEFORMATEX  pWaveFormat,
  ULONG          pktSizeMs
);
```

Parameters

hStream

Stream handle returned by EpGetStreamHandle.

pWaveFormat

Pointer to a WAVEFORMATEX data structure which contains codec information.

pktSizeMs

Packet size in milliseconds. If value 0 (zero) is specified a default value (20) is used.

Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

EpApiTraceLevelSet

Sets EpApi (Rtp Library) trace level.

Syntax

```
CMAPI bool EpApiTraceLevelSet (
  int  traceLevel
);
```

Parameters

traceLevel

Required Rtp Library trace level. This parameter can be one of the following values:

Table 23: Error Codes for EpApiTraceLevelSet

Trace level	Description
0 -Error	Output only error messages (reported in Windows Event Log).
1 -Alarm	Output alarms and error messages (reported in Windows Event Log).
2 -Warning	Output warnings, alarms and error messages (reported in Windows Event Log).
3 -Info	Output informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).
4 -Debug	Output debug information, informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).

Return Value

Current trace level.

EpApiGetLastError

Retrieves last-error code value. The last-error code is maintained on a per-thread basis.

Syntax

CMAPI int EpApiGetLastError();

Parameters

This function has no parameters.

Return Value

The return value is the calling thread's last error code.

EpApi Error Codes

Most of the EpApi functions do not return a specific cause of an error when the function returns but rather set global error code value which can be retrieved by calling EpApiGetLastError function. The following list describes possible error codes returned by EpApiGetLastError function. Errors are listed in numerical order.

Table 24: Error Codes for EpApi

Return code/Value	Description
EP_ERR_OK 0	No error occurred.
EP_ERR_INIT 17002	EpAPI is not initialized.
EP_ERR_PARAM_INVALID 17003	Invalid parameter.
EP_ERR_ADDR_NOTAVAIL 17100	Unable to create endpoint with specified IP address family
EP_ERR_ADDR_INUSE 17101	Address (Protocol -IP address -port) is already in use.
EP_ERR_HANDLE_INVALID 17102	Invalid endpoint or stream handle.

Callback Function

An application can define a callback function in order to receive information about such things as operation completions, data transfers, and errors. Callback functions can be specified when an endpoint is created, when a stream callback is opened, and when a stream callback operation is initiated. If a callback operation is not specified a corresponding stream callback is invoked, if defined. If a stream callback is not specified a corresponding callback endpoint is invoked, if defined.

Note If the callback function is defined, it is invoked for every operation that is initiated on a corresponding stream, endpoint, etc. Consideration should be given to the case where a callback function is defined as a method in an object that is dynamically created and destroyed. In that case destruction should not occur until all initiated operations are complete.

Endpoint Callback

Syntax

```
typedef void (WINAPI  *PRTPENDPOINTCALLBACK) (
  HANDLE           hEp,
  HANDLE           hStream,
  DWORD            dwError,
  PUCHAR           pData,
  DWORD            dwDataSize,
  LPVOID           pUserData,
  bool             bIsSilence,
  StreamDirection  streamDir
);
```

Parameters

hEp

Endpoint handle

hStream

Rtp stream handle

dwError

If not 0 (zero), indicates an error

pData

Endpoint handle

dwDataSize

Number of bytes received / transferred.

pUserData

Application data associated with an operation.

bIsSilence

If set to true, indicates that silence has been detected.

streamDir

Stream direction. Can be one of the following:

- ToApp
- ToNwk

Data Callback

Syntax

```
typedef void (WINAPI  *PRTPDATACALLBACK) (
  HANDLE           hStream,
  DWORD            dwError,
  PUCHAR           pData,
  DWORD            dwDataSize,
  LPVOID           pUserData,
  bool             bIsSilence,
);
```

Parameters

hStream

Rtp stream handle

dwError

If not 0 (zero), indicates an error

pData

Endpoint handle

dwDataSize

Number of bytes received / transferred.

pUserData

Application data associated with an operation.

bIsSilence

If set to true, indicates that silence has been detected.

Data Structures

RTPADDR

Basic endpoint data structure which contains all endpoint related data, such as IP address, UDP port number, etc.

```
typedef struct sRTPAddrInfo {
  ADDRINFOT         info,
  SOCKADDR_STORAGE  addr,
  SOCKET            sock,
  bool              multicast,
  DWORD             dscp,
  SRTPINFO2         srtp,
  RTPSIL            silence,
  ULONG             pktSizeMs
} RTPADDR, *PRTPADDR;
```

Where:

info

System defined ADDRINFOT structure

addr

System defined SOCKADDR_STORAGE structure

sock

System defined SOCKET data (bound socket)

multicast

If set to true, RTPADDR instance represents multicast address, otherwise unicast.

dscp

DSCP / QoS data

srtp

SRTP data

silence

Silence processing parameters

pktSizeMs

Packet size in milliseconds

RTPSIL

Contains silence processing related data for a specific endpoint. It uses the following SilenceType enumeration:

```
typedef enum {
  Off     = 0,
  Packets = Off + 1,
  Energy  = Packets +1
} SilenceType;

typedef struct {
  SilenceType  type,
  ULONG        duration,
  ULONG        threshold,
  ULONG        currentOffset,
  bool         detecting
} RTPSIL, *PRPTSIL;
```

Where:

type

Silence detection type as it is defined in SilenceType.

duration

Duration in milliseconds.

threshold

Energy threshold.

currentOffset

Silence offset (G.729).

detecting true

Silence detection enabled.

RTPCODEC

```
typedef struct {
    WAVEFORMATEX   wfe;
    WORD           (WINAPI *formatTag) ();
    WORD *         (WINAPI *supported) (ULONG & nmb);
    ULONG          (WINAPI *fmtBytesToThis) (WORD fmt, ULONG len);
    ULONG          (WINAPI *thisBytesToFmt) (ULONG len, WORD fmt);
    UCHAR          (WINAPI *pad) ();
    PXLATE          xlateTo;
    PXLATE          xlateFrom;
    PRTPSIL        (WINAPI *silenceInit)(PRTPSIL ps, SilenceType type,
 ULONG duration, ULONG threshold);
    ULONG          (WINAPI *silenceSet) (PRTPSIL, PUCHAR, ULONG);
    bool           (WINAPI *isSilence)  (PRTPSIL, ULONG pcktSizeInMs,
 bool & beenChanged, PUCHAR, ULONG);
    void           (WINAPI *silenceFree)(PRTPSIL);
} RTPCODEC, *PRTPCODEC;
```

Trace Options

Rtp Library have several logging options to facilitate application debugging and trouble-shooting:

Reporting in the Windows Event Log

Sending trace data to the OutputDebugString and can be view by any “trace listener”, for example Sysinternals DebugView

Providing trace data to an application in the trace callback

Trace Level

Trace level specifies what messages are to be included in trace output and is defined as follows:

Trace level	Description
0 -Error	Output only error messages (reported in Windows Event Log).
1 -Alarm	Output alarms and error messages (reported in Windows Event Log).
2 -Warning	Output warnings, alarms and error messages (reported in Windows Event Log).
3 -Info	Output informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).
4 -Debug	Output debug information, informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).

Trace level can be set and modified with the EpApiTraceLevelSet() function.

Trace Callback Function

An application can set trace callback. The callback function will be invoked by Rtp Library whenever it is ready to record a trace and will be provided with the trace record data. Trace callback can be set and modified when EpApi is initialized. Trace callback function type is declared as follows:

Syntax

```
typedef void (WINAPI  *PRTPLIBTRACE) (
  int           level,
  const         _TCHAR  *pData
);
```

Parameters

level

Current trace record level.

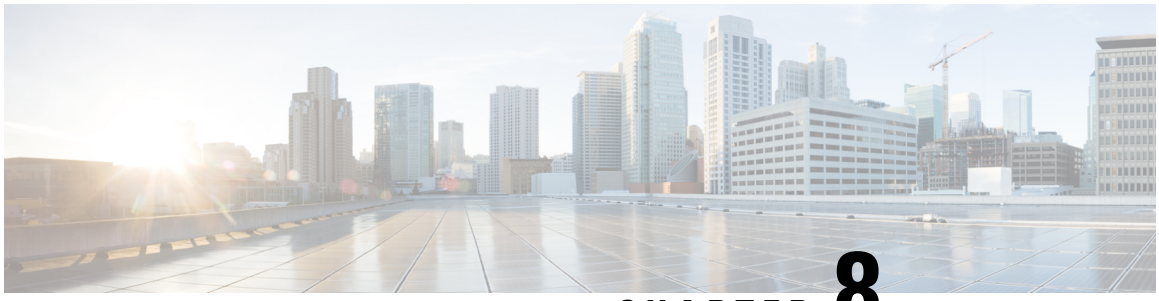
pData

Pointer to the current trace record data.

Known Problems or Limitations

Below is the list of currently known Rtp Library problems and limitations:

- CSCsy13584 – RtpLib: The only supported PCM encoding is 8k16bit, mono
- There is no G.729 transcoding available



CHAPTER 8

Cisco Unified TAPI Examples

This chapter provides examples that illustrate how to use the Cisco Unified TAPI implementation. This chapter includes the following subroutines:

MakeCall

```
STDMETHODIMP CTACtrl::MakeCall(BSTR destNumber, long pMakeCallReqID,
 long hLine, BSTR user2user, long translateAddr) {
  AFX_MANAGE_STATE(AfxGetStaticModuleState())

  USES_CONVERSION;
  tracer->tracef(SDI_LEVEL_ENTRY_EXIT, "CTACtrl::Makecall %s %d %d %s %d\n",
    T2A((LPTSTR)destNumber), pMakeCallReqID, hLine, T2A((LPTSTR)user2user),
    translateAddr);

  //CtPhoneNo   m_pno;
  CtTranslateOutput   to;

  //LPCSTR  pszTranslatable;
  CString sDialable;

  CString theDestNumber(destNumber);

  CtCall* pCall;
  CtLine* pLine = CtLine::FromHandle((HLINE)hLine);

  if (pLine = = NULL) {
    tracer->tracef(SDI_LEVEL_ERROR, "CTACtrl::MakeCall : pLine = = NULL\n");
    return S_FALSE;
} else {
    pCall = new CtCall(pLine);
    pCall->AddSink(this);

    sDialable = theDestNumber;

    if (translateAddr) {
      //m_pno.SetWholePhoneNo((LPCSTR)theDestNumber);
      //pszTranslatable = m_pno.GetTranslatable();
        if (TSUCCEEDED(to.TranslateAddress(pCall->GetLine()->GetDeviceID(),
```

```
          (LPCSTR)theDestNumber)) ) {
            sDialable = to.GetDialableString();
        }
    }
    TRESULT tr = pCall->MakeCall((LPCSTR)sDialable, 0, this);
    if( TPENDING(tr) || TSUCCEEDED(tr)) {
      //GCGC the correct hCall pointer is not being returned yet
      if (translateAddr)
        Fire_MakecallReply(hLine, (long)tr, (long)pCall->GetHandle(),
          sDialable.AllocSysString());
      else
       Fire_MakecallReply(hLine, (long)tr, (long)pCall->GetHandle(),destNumber);


      return S_OK;
    } else {
      //GCGC delete the call that was created above.
      tracer->tracef(SDI_LEVEL_ERROR, "CTACtrl::MakeCall : pCall->MakeCall
failed\n");
      delete pCall;
      return S_FALSE;
    }
  }
}
```

OpenLine

```
STDMETHODIMP CTACtrl::OpenLine(long lDeviceID, BSTR lineDirNumber,
 long lPriviledges,    long lMediaModes, BSTR receiveIPAddress,
 long lreceivePort) {
 USES_CONVERSION;
 tracer->tracef(SDI_LEVEL_ENTRY_EXIT, "CTACtrl::OpenLine %d %s %d %d %s %d\n",
     lDeviceID, T2A((LPTSTR)lineDirNumber), lPriviledges, lMediaModes,
     T2A((LPTSTR)receiveIPAddress), lreceivePort);

 int lineID;
 TRESULT tr;
 CString strReceiveIP(receiveIPAddress);
 CString strReqAddress(lineDirNumber);

 //bool bTermMedia = ((!strReceiveIP.IsEmpty()) && (lreceivePort! = 0));
 bool bTermMedia = (((lMediaModes & LINEMEDIAMODE_AUTOMATEDVOICE) ! = 0)  &&
                  (lreceivePort! = 0) && (!strReceiveIP.IsEmpty()));
 CtLine* pLine;

 AFX_MANAGE_STATE(AfxGetStaticModuleState())

 tracer->tracef(SDI_LEVEL_DETAILED, "TAC: --> OpenLine()\n");

 if ((lDeviceID<0) && !strcmp((char *)lineDirNumber, "")) {
  tracer->tracef(SDI_LEVEL_ERROR, "TCD: error -bad device ID and no dirn to
open\n");
  return S_FALSE;
 }
 lineID = lDeviceID;

 if (lDeviceID<0) {
  //search for line ID in list of lines.
  CtLineDevCaps   ldc;
```

```
 int numLines = ::TfxGetNumLines();
 for( DWORD nLineID = 0; (int)nLineID < numLines; nLineID++ ) {
  if( /*ShouldShowLine(nLineID) &&*/ TSUCCEEDED(ldc.GetDevCaps(nLineID)) ) {
   CtAddressCaps ac;
   tracer->tracef(SDI_LEVEL_DETAILED, "CTACtrl::OpenLine :
                    Calling ac.GetAddressCaps %d 0\n", nLineID);
   if ( TSUCCEEDED(ac.GetAddressCaps(nLineID, 0)) ) {
    // GCGC only one address supported
    CString strCurrAddress(ac.GetAddress());
    if (strReqAddress = = strCurrAddress) {
     lineID = nLineID;
     break;
    }
   }

  } else {
   tracer->tracef(SDI_LEVEL_ERROR, "TAC: error -GetAddressCaps() failed\n");
  }
 }
}

if (lDeviceID<0) {
 tracer->tracef(SDI_LEVEL_ERROR,
    "TAC: error -could not find dirn %s to open line.\n",(LPCSTR)lineDirNumber);

 return S_FALSE;
}

// if we are to do media termination; negotiate the extensions version

DWORD retExtVersion;
if (bTermMedia) {
 TRESULT tr3;
 tracer->tracef(SDI_LEVEL_DETAILED,
     "TAC: lineNegotiateExtVersion -appHandle = %d, deviceID = %d, API ver = %d,

                    HiVer = %d, LoVer = %d\n", CtLine::GetAppHandle(), lineID,
                    CtLine::GetApiVersion(lineID),
                    0x80000000 | 0x00010000L,
                    0x80000000 | 0x00020000L );
 tr3 = ::lineNegotiateExtVersion(CtLine::GetAppHandle(),
                    lineID, CtLine::GetApiVersion(lineID),
                    0x80000000 | 0x00010000L,    // TAPI v1.3,
                    0x80000000 | 0x00020000L,
                    &retExtVersion);
 tracer->tracef(SDI_LEVEL_DETAILED,
                    "TAC: lineNegotiateExtVersion returned: %d\n", tr3);
}

pLine = new CtLine();
tr = pLine->Open(lineID, this, lPriviledges, lMediaModes);
if( TSUCCEEDED(tr)) {
 if (bTermMedia) {
  if (retExtVersion = = 0x10000) {
   CiscoLineDevSpecificUserControlRTPStream dsucr;
   dsucr.m_RecievePort = lreceivePort;
   dsucr.m_RecieveIP = ::inet_addr((LPCSTR)strReceiveIP);
   TRESULT tr2;

   tr2 = ::lineDevSpecific(pLine->GetHandle(),
                        0,0, dsucr.lpParams(),dsucr.dwSize());
   tracer->tracef(SDI_LEVEL_DETAILED,
                    "TAC: lineDevSpecific returned: %d\n", tr2);
  } else {
```

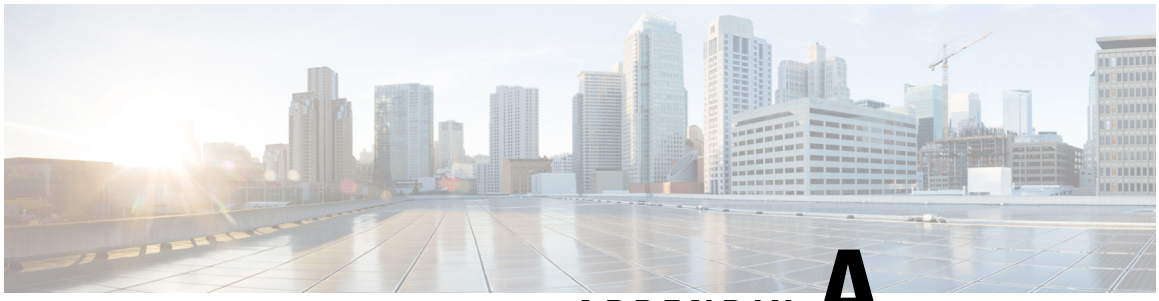
```
    //GCGC here put in the new calls to set the media types!
    CiscoLineDevSpecificUserControlRTPStream2 dsucr;
    dsucr.m_RecievePort = lreceivePort;
    dsucr.m_RecieveIP = ::inet_addr((LPCSTR)strReceiveIP);
    dsucr.m_MediaCapCount = 4;

    dsucr.m_MediaCaps[0].MediaPayload = 4;
    dsucr.m_MediaCaps[0].MaxFramesPerPacket = 30;
    dsucr.m_MediaCaps[0].G723BitRate = 0;
    dsucr.m_MediaCaps[1].MediaPayload = 9;
    dsucr.m_MediaCaps[1].MaxFramesPerPacket = 90;
    dsucr.m_MediaCaps[1].G723BitRate = 1;
    dsucr.m_MediaCaps[2].MediaPayload = 9;
    dsucr.m_MediaCaps[2].MaxFramesPerPacket = 90;
    dsucr.m_MediaCaps[2].G723BitRate = 2;
    dsucr.m_MediaCaps[3].MediaPayload = 11;
    dsucr.m_MediaCaps[3].MaxFramesPerPacket = 90;
    dsucr.m_MediaCaps[3].G723BitRate = 0;

    TRESULT tr2;

    tr2 = ::lineDevSpecific(pLine->GetHandle(),
                            0,0, dsucr.lpParams(),dsucr.dwSize());
    tracer->tracef(SDI_LEVEL_DETAILED,
                     "TAC: lineDevSpecific returned: %d\n", tr2);
   }
  }

  CtAddressCaps ac;
  LPCSTR  pszAddressName;
  if ( TSUCCEEDED(ac.GetAddressCaps(lineID, 0)) ) {
   // GCGC only one address supported
   pszAddressName = ac.GetAddress();
  } else {
   pszAddressName = NULL;
   tracer->tracef(SDI_LEVEL_ERROR, "TAC: error -GetAddressCaps() failed.\n");
  }

  OpenedLine((long)pLine->GetHandle(), pszAddressName, 0);

  // now let's try to open the associated phone device
  // Get the phone from the line

  DWORDnPhoneID;
  bool b_phoneFound = false;
  CtDeviceID  did;
  if((m_bUsesPhones) && TSUCCEEDED(did.GetID("tapi/phone", pLine->GetHandle()))
) {
   nPhoneID = did.GetDeviceID();
   tracer->tracef(SDI_LEVEL_DETAILED,
                    "TAC: Retrieved phone device %d for line.\n",nPhoneID);

   // check to see if phone device is already open

   long hPhone;
   CtPhone* pPhone;
   if (!m_deviceID2phone.Lookup((long)nPhoneID,hPhone)) {
    tracer->tracef(SDI_LEVEL_SIGNIFICANT,
                   "TAC: phone device not found in open list, opening it...\n");

    pPhone = new CtPhone();
    TRESULT tr_phone;
    tr_phone = pPhone->Open(nPhoneID,this);
    if (TSUCCEEDED(tr_phone)) {
```

```
     ::phoneSetStatusMessages(pPhone->GetHandle(),
         PHONESTATE_DISPLAY | PHONESTATE_LAMP |
         PHONESTATE_HANDSETHOOKSWITCH | PHONESTATE_HEADSETHOOKSWITCH |
         PHONESTATE_REINIT | PHONESTATE_CAPSCHANGE | PHONESTATE_REMOVED,
         PHONEBUTTONMODE_KEYPAD | PHONEBUTTONMODE_FEATURE |
         PHONEBUTTONMODE_CALL |
         PHONEBUTTONMODE_LOCAL | PHONEBUTTONMODE_DISPLAY,
         PHONEBUTTONSTATE_UP | PHONEBUTTONSTATE_DOWN);
     m_phone2line.SetAt((long)pPhone->GetHandle(), (long)pLine->GetHandle());
     m_line2phone.SetAt((long)pLine->GetHandle(),(long)pPhone->GetHandle());
     m_deviceID2phone.SetAt((long)nPhoneID,(long)pPhone->GetHandle());
     m_phoneUseCount.SetAt((long)pPhone->GetHandle(), 1);
    } else {
     tracer->tracef(SDI_LEVEL_ERROR,
                       "TAC: error -phoneOpen failed with code %d\n", tr_phone);
    }
   } else {
    pPhone = CtPhone::FromHandle((HPHONE)hPhone);
    long theCount;

    if (m_phoneUseCount.Lookup((long)pPhone->GetHandle(),theCount))
     m_phoneUseCount.SetAt((long)pPhone->GetHandle(), theCount+1);
    else {
     //GCGC this would be an error condition!
     tracer->tracef(SDI_LEVEL_ERROR,
        "TAC: error -m_phoneUseCount does not contain phone entry.\n");
    }
   }
  } else {
   tracer->tracef(SDI_LEVEL_ERROR,
                     "TAC: error -could not retrieve PhoneID for line.\n");
  }
  tracer->tracef(SDI_LEVEL_DETAILED, "TAC: <--OpenLine()\n");
  return S_OK;
 } else {
  tracer->tracef(SDI_LEVEL_ERROR, "TAC: error -lineOpen failed: %d\n", tr);
  tracer->tracef(SDI_LEVEL_DETAILED, "TAC: <--OpenLine()\n");
  OpenLineFailed(tr,0);
  delete pLine;
  return S_FALSE;
 }
}
```

CloseLine

```
STDMETHODIMP CTACtrl::CloseLine(long hLine) {
 AFX_MANAGE_STATE(AfxGetStaticModuleState())

 tracer->tracef(SDI_LEVEL_ENTRY_EXIT, "CTACtrl::CloseLine %d\n", hLine);

 CtLine* pLine;
 pLine = CtLine::FromHandle((HLINE) hLine);

 if (pLine! = NULL) {
  // close the line
  pLine->Close();
  // remove it from the list
  delete pLine;
  long hPhone;
```

```
  long theCount;
  if ((m_bUsesPhones) && (m_line2phone.Lookup(hLine,hPhone))) {
   CtPhone* pPhone = CtPhone::FromHandle((HPHONE)hPhone);
   if (pPhone! = NULL) {
    if (m_phoneUseCount.Lookup(hPhone,theCount))
     if (theCount>1) {
      // decrease the number of lines using this phone
      m_phoneUseCount.SetAt(hPhone,theCount-1);
     }
     else {
      //nobody else is using this phone, so let's remove it.
      m_deviceID2phone.RemoveKey((long)pPhone->GetDeviceID());
      m_phone2line.RemoveKey(hPhone);
      m_phoneUseCount.RemoveKey(hPhone);

      //now let's close the phone
      pPhone->Close();
     }
   }
   //either way, remove the map entry from line to phone.
   m_line2phone.RemoveKey(hLine);
  }
  return S_OK;
 }
 else
  return S_FALSE;
}
```



APPENDIX A

Message Sequence Charts

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

This appendix contains the following sections:

Abbreviations

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

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

3XX

Application monitors B.

Table 25: 3XX

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

Agent Greeting

Configuration

Customer Phone—IP Phone A with DN 1001.

Agent Phone—IP Phone B with DN 1002.

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

Supervisor Phone—IP Phone D with DN 1003.

IVR1—with DN 5555

IVR2—with DN 6666

Procedure

Application monitoring all lines on all devices.

New extension is negotiated when application opens lines.

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

Table 26: StartSendMediaToBIB Success Case

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

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

Table 27: StopSendMediaToBIB Success Case

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

Table 28: StartSendMediaToBIB Failure While Monitoring in Progress at Agent Side

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

Table 29: StartSendMediaToBIB Followed by Monitoring Request

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

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

Table 30: StartSendMediaToBIB While Recording Is in Session

Action	Events, requests and responses
Make call from 1001 to 1002, and 1002 answers	At 1001: CONNECTED Calling = 1001 Called = 1002 Connected = 1002 At 1002: CONNECTED Calling = 1001 Called = 1002 Connected = 1001
Application sends CCiscoLineDevSpecificStartCallRecording to 1002	At 1002: CCiscoLineDevSpecificStartCallRecording will be successful and recording is in session

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

Table 31: StartSendMediaToBIB Followed by Recording Request

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

Action	Events, requests and responses
Application sends CCiscoLineDevSpecificStartCallRecording to 1002	At 1002: CCiscoLineDevSpecificStartCallRecording will be successful and recording is in session

Table 32: StartSendMediaToBIB Failure While Barge in Session

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

Table 33: StartSendMediaToBIB Followed by Barge From Shared Line

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

Table 34: This Behavior Is Also Seen During Consult Operation. Agent Holds Call While Agent Greeting Is Being Played

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

Action	Events, requests and responses
1002 put call on hold	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event Call will go on hold With StopReception and StopTransmission event At 5555: Call goes IDLE
1002 Unhold scenario	At 1002: Call will go CONNECTED with StartTransmission and StartReception.

Table 35: Agent Redirects Call While Agent Greeting Is Being Played

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

Action	Events, requests and responses
Application issues CCiscoLineDevSpecificStartSendMediaToBIBRequest on 1002 (CM feature creates server call to IVR1 5555, 5555 answers call) Server-IVR call is redirected to BIB IVR1 selects/plays agent's greeting	At 1002: the request is successful Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_STARTED) event At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = CgpnToIVR Called = 5555 Connected = CgpnToIVR CallAttributeBitMask = ServerCall bit will be set At 5555: CONNECTED, dwreason = LINECALLREASON_UNKNOWN (unknown) ExtendedCallReason = CtiReasonSendMediaToBIB Calling = 5555 Called = 5555 Connected = CallAttributeBitMask = ServerCall bit will be set Media event sent to application (StartTransmissionEvent)
Application redirects call on 1002 to 1003	At 1003: New call from 1002 At 1002: Call goes IDLE No MEDIA_TO_BIB_ENDED event At 5555: Call goes IDLE

Table 36: IVR1 Redirects Call to IVR2

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

Action	Events, requests and responses
Application redirect call on IVR1 to IVR2 IVR2 answers and plays second agent greeting	At 5555: Call goes IDLE At 6666: Calling = Called = 6666 Connected = Redirecting = 5555 Redirection = 6666 CallAttributeBitMask = BIBCall (StartTransmissionEvent)
IVR2 drops call after agent greeting completes	At 1002: Application receives LineCallDevSpecific (SLDSMT_MEDIA_TO_BIB_ENDED,0,0) event At 6666: Call goes IDLE

Table 37: Application-2 Opened Line After Agent Greeting Is in Playing

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

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

Table 38: Start Agent Greeting After Conference Is Setup

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

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

Agent Zip Tone

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

Configuration

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

All Lines are Opened with Ext Version – 0x000B0000

Table 39: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – Remote

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

Table 40: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – Local

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

Table 41: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent. PlayToneDirection – BothLocalandRemote/NoLocalOrRemote

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

Table 42: Application Issues the Play Tone Request (with Unsupported Tone) When the Call Is Established Between Customer and Agent. PlayToneDirection – Local

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

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

Configuration

A (Customer/Remote) is SCCP Phone.

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

Table 43: Application Issues the Play Tone Request on a CTI Port with PlayToneDirection – Local/Remote

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

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

Configuration

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

Table 44: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent (Shared Line). PlayToneDirection – Local

Action	Expected events
LineInitialize. LineOpen on A, B, B' The CallToneChangedEvent message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B and B'. A calls B;B and B' starts ringing; B answers the Call B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input. Variants: B' issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote. B issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote. A issues the LineDevSpecific (start PlayTone) request with agent callid and ZIP Tone as input direction remote.	

Action	Expected events
	Zip Tone is played at B. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local)) is fired for B indicating Zip Tone has been played on B. There is no Zip Tone played at B'and no Zip tone notification on B'. The LineDevSpecific (start PlayTone) request fails with Error LINEERR_OPERATIONFAILED Zip Tone is played at A. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local))) will be fired for A also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 1(remote) will be fired for B. There is no Zip Tone played at B'and no Zip tone notification on B'. Zip Tone is played at B and B'. Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 = CTONE_ZIP, dwParam3 = 0(local))) is fired for B and B' also Line_DevSpecific (dwparam1 = SLDSMT_CALL_TONE_CHANGED, dwParam2 =

Action	Expected events
	CTONE_ZIP, dwParam3 = 1(remote) is fired for A.

Table 45: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent (Intercom Line). PlayToneDirection – Local

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

Conference Scenario: PlayToneDirection -local.

Configuration

A, B, and C are SCCP Phones.

Table 46: Conference Scenario. PlayToneDirection – Local

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

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

Configuration

A and B are SCCP Phones.

Table 47: Application Issues the Play Tone Request When the Call Is Established Between Customer and Agent, Agent Puts the Call on Hold. PlayToneDirection – Remote

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

Announcement Call

Prerequisites

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

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

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

Announcement with ID "WelcomeID" is defined on CUCM.

Table 48: Create Announcement Call

Action	TAPI Messages	TAPI Structure

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

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

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

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

Announcement with ID "WelcomeID" is defined on CUCM.

Table 49: Drop Announcement Call

Action	TAPI Messages	TAPI Structures

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

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

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

Precondition: No Persistent call on CTIRD-1

Table 50: Negative -Create Announcement Call Failed / No Persistent Call

Action	TAPI Messages	TAPI Structures

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

Blind Transfer

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

Table 51: Message Sequences for Blind Transfer

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

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

Call Control Discovery

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

Configuration

SCCP phone A(1900) are registered to cluster A

Phones A are associated with the end-user cluster1

SCCP phone B(1000) registered to cluster B

Phones B are associated with the end-user cluster2

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

Procedure

Application monitors A

Application sends a lineMakeCall at A to call B

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

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

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

Configuration

SCCP phone A are registered to cluster A.

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

SCCP phone B(1000) registered to cluster B.

Phones B are associated with the end-user “cluster2”.

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

Procedure

Application monitors A.

Application sends a lineMakeCall at A to call B.

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

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

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

Configuration

SCCP phone A and B are registered to cluster A.

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

SCCP phone C(1000) registered to cluster B.

Phones C are associated with the end-user cluster2.

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

Procedure

Application monitors A and B.

Application sends a lineMakeCall at A to call B

Table 52: Basic Call Initiated From TAPI From Phone A(1900) and B(1901) on Cluster 1, B Redirects to Phone C(1000) on Cluster2 with PSTN Failover Rule Set

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

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

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

Configuration

SCCP phone A and B are registered to cluster A.

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

SCCP phone C(1000) registered to cluster B.

Phones C are associated with the end-user cluster2.

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

Procedure

Application monitors A and B.

Application sends a lineMakeCall at A to call B.

Table 53: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Transfers to Phone C(1000) on Cluster 2 with PSTN Failover Rule

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

Action	CTI messages	TAPI messages
B(1901) setups transfer to C(1000) This call first will be intercepted by CCD Requesting Feature, and CCD Requesting feature will extend this call to SIP trunk SIP trunk rejects this call due to no more bandwidth available CCD Requesting feature will start PSTN failover by directing this caller to 1000's PSTN failover number, i.e. 14089721000. Call is sent out to a PSTN GW. TSPI_linegetcallinfo on Consult call on B with C. B completes transfer		

Action	CTI messages	TAPI messages
		B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) LINECALLINFO.dwCallID = 0x00400BBA LINECALLINFO.dwOrigin = 0x00000001 LINECALLINFO.dwReason = 0x00000001 LINECALLINFO.dwCallerID = 1901(B) LINECALLINFO.dwCallerIDName = LINECALLINFO.dwCalledID = 1000: LINECALLINFO.dwCalledIDName = CCD Pattern LINECALLINFO.dwConnectedID = 1000(C) LINECALLINFO.dwConnectedIDName = LINECALLINFO.dwRedirectionID = 1000 LINECALLINFO.dwRedirectionIDName = LINECALLINFO.dwRedirectingID = 1000 LINECALLINFO.dwRedirectingIDName = CCD Pattern Extendedcallreason = CtiReasonSAF_CCD_PSTNFailover B: LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED) ExtendCallReason =

Action	CTI messages	TAPI messages
		CtiReasonTransferredCall

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

Configuration

SCCP phone A and B are registered to cluster A

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

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

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

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 54: Call Initiated From TAPI From Phone A and B on Cluster 1, B Sets Up Conference to Phone C(1000) on Cluster 2 with PSTN Failover Rule

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

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

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

Configuration

SCCP phone A(1900) are registered to cluster A

Phones A are associated with the end-user cluster1

SCCP phone B(1000) registered to cluster B

Phones B are associated with the end-user cluster2

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

Procedure

Application monitors A

Application sends a lineMakeCall at A to call B

Table 55: Basic Call Initiated From TAPI From Phone A on Cluster 1 to Phone B on Cluster 2 Over SAF Trunk

Action	CTI messages	TAPI messages
A dials 1000	A receives NewCallEvent and CallStateChangeEvent (Dialtone/Dialing)	A: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING) CallerID = A / CalledID = “”
	A receives CallStateChangeEvent (PROCEEDING)	LineA: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING)/ LINE_CALLINFO CallReason = LINECALLREASON_DIRECT CallerID = A / CalledID = 1000 / ConnectedID = / RedirectingID = / RedirectionID =

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

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

Configuration

SCCP phone A and B are registered to cluster A

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

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

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

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 56: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Redirects to Phone C(1000) on Cluster 2 Over SAF Trunk

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

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

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

Configuration

SCCP phone A and B are registered to cluster A

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

SCCP phone C(1000) registered to cluster B

Phones C are associated with the end-user cluster2

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

Procedure

Application monitors A and B

Application sends a lineMakeCall at A to call B

Table 57: Basic Call Initiated From TAPI From Phone A and B on Cluster 1, B Transfers to Phone C(1000) on Cluster 2 Over SAF Trunk

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

Action	CTI messages	TAPI messages
B setup transfers to C(1000), through the ICT(SAF) trunk	B: receives CPIC event	B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DIALTONE/ LINECALLSTATE_DIALING/Proceeding) LineB: LINE_CALLSTATE (LINECALLSTATE_RINGBACK / LINECALLSTATE_CONNECTED) CallReason = LINECALLREASON_DIRECT
Complete transfer on B		B: LINE_APPNEWCALL, LINE_CALLSTATE (LINECALLSTATE_DISCONNECTED)
TSPI_linegetcallinfo on disconnected call on B		ExtendCallReason = CtiReasonTransferredCall

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

CallFwdAll Notification

This section describes the CallFwdAll Notification usecases.

Application Pressed CFwdAll on TAPI Monitored Device

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

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

TAPI Monitored Device Goes Off Hook

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

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

Application Monitors Off Hook Device

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

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

Application Monitors Device After User Presses CFwdAll

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

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

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

User Presses CFwdAll Softkey After Device Is Off Hook

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

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

User Presses CFwdAll Softkey on a Multiline Device

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

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

User Presses CFwdAll on a Multiline Device by Selecting a Line

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

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

Shared Line Scenario on Pressing CFwdAll Softkey

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

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

Cancellation of CFwdAll

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

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

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

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

Calling Party IP Address

Basic Call

TAPI application monitors party B

Party A represents an IP phone

A calls B

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

Consultation Transfer

TAPI application monitors party C

Party B represents an IP phone

A talks to B

B initiates a consultation transfer call to C

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

B Completes the transfer

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

Consultation Conference

TAPI application monitors party C

Party B represents an IP phone

A talks to B

B initiates a consultation conference call to C

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

B Completes the conference

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

Redirect

TAPI application monitors party B and party C

Party A represents an IP phone

A calls B

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

Party A redirects B to party C

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

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

Calling Party Normalization

Incoming Call From PSTN to End Point

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

Incoming Call From National PSTN to CTI-Observed End Point

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

Incoming Call From International PSTN to CTI-Observed End Point

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

Outgoing Call From CTI-Observed End Point to PSTN Number

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

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

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

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

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

Call PickUp

Registering CallPickUpGroup for Notification

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

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

Pickup group P1:1111 is configured

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

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

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

UnRegistering CallPickUpGroup for Notification

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

Re-Registering CallPickUpGroup for Notification

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

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

Registering/UnRegistering CallPickUpGroup for Notification with Invalid Information

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

CallPickUp After Enabling Auto Call Pickup Enabled

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

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

CallPickUp with Auto Call Pickup Enabled Disabled

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

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

CallPickUp with Multiple Calls Available

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000	CallPickUp Group DN and Partition Information will be sent to application
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
P1:4001 calls P1:1001	Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
LineGetCallInfo on Call	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information
LineGetCallInfo on Call2	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4001 dwCalledID : 1001 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information

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

CallPickupGroup Changed for a Device on AdminPage

Pickup group P1:9999 is configured

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

CallPickUpGroup Partition or DN Information Updated

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

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

CallPickUpGroup Is Deleted

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

Call Queuing

HP1 is a Huntpilot with the below configuration:

"Queue Calls" check box is selected.

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

HP1: LG1

HP2: LG1

A, B (IP phones/CTI Ports)

Table 58: Basic Hunt List Call (HP1 Has at Least One Member Free)

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

Table 59: Basic Hunt List Call. HP1 Has All Members Busy (LG1)

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

Action	Expected events

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

Action	Expected events
Display Line Group Member DN as Connected Party is enabled	Same as above

Table 60: Hunt List Call to HP1 When Queue Depth Is Reached. (Maximum Number of Callers Allowed in Queue = 2)

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

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

Action	Expected events
Variance: Destination When Queue is Full = HP2 Call on LG1 of HP2 goes idle (LG1 is free). Queued call from A is de-queued and offered on LG1.	

Action	Expected events
	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 of HP2 HuntPilot = HP2 At LG1 of HP2: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x30(CallDeQueueAgentsBusy) Caller = A Called = HP1

Action	Expected events
	Connected = A

Table 61: Hunt List Call to HP1 and Maximum Wait Time in Queue Is Met

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

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

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

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

Action	Expected events
	Called = HP1 Connected = A

Table 62: Hunt List Call to HP1 and No Agents Logged In or Registered

Action	Expected events
App initiates call from A to HP1. (None of the Huntmembers are registered or logged in). Destination When There Are No Agents Logged In or Registered = ' B' Call offered on B. B Answers the call.	

Action	Expected events
	At A: LINE_CALLSTATE -RINGBACK CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 At B: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A, Called = HP1 At A: LINE_CALLSTATE -CONNECTED CallReason = x1(Direct) ExtendedCallReason = x1(DirectCall) Caller = A Called = HP1, HuntPilot = HP1 Connected = B HuntPilot = At B: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x31(CallDeQueueAgentsUnavailable) Caller = A Called = HP1 Connected = A

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

Table 63: Basic Hunt List Call. A Calls B, and B Redirects/forwards/transfers the Call to HP1

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

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

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

Action	Expected events
	Connected = A HuntPilot =

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

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

Action	Expected events
	HuntPilot = Connected = LG1 HuntPilot =

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

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

Action	Expected events
	Caller = A, Called = B Connected = LG1

Table 64: Basic Hunt List Call. HP1 Has All Members Busy (LG1), Queued Call on A Is Redirected

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

Action	Expected events
Queued Call on A is redirected to B. B Answers. Call on LG1 goes idle (LG1 is free). Queued call from B is de-queued and offered on LG1. LG1 Answers the call.	

Action	Expected events
	At B: LINE_CALLSTATE -CONNECTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x6(Redirect) Caller = HP1 Called = B, HuntPilot = Connected = HP1 HuntPilot = At LG1: LINE_CALLSTATE -ACCEPTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = B, Called = HP1 HuntPilot = HP1 At B: LINE_CALLSTATE -CONNECTED CallReason = x40(LINECALLREASON_REDIRECT) ExtendedCallReason = x2e(CallDeQueue) Caller = B Called = B HuntPilot = Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED CallReason = x400(unknown) ExtendedCallReason = x2e(CallDeQueue) Caller = B Called = HP1

Action	Expected events
	HuntPilot = HP1 Connected = B

Table 65: Hunt List Call to HP1 and No Agents Logged In or Registered

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

FailOver or FailBack Scenario

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

Action	Expected events
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information
Stop Primary CTI Manager	OutofService for the line P1:1111 INService for the line P1:1111. **Note** There will not be any notification for the existing calls.

GroupCallPickup

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

Pickup group P1:1111 is configured and opened

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

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

P1:4000 and P1:4001 are configured

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

Action	Expected
LineGetCallInfo	LINE_CALLINFO dwCallState : PickupCallState (0x10000000) dwCallerId : 4000 dwCalledID : 1002 dwCallorigin: Internal dwCallReason : Direct Check for all fields of Calling and Called Information
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupCallPickup option and GroupPickUp DN 1111 on P1:2000	Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupCallPickup option and GroupPickUp DN 1111 on P1:2000Events on P1:2000: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4000, Called = 1111, Connected = 4000, dwReason = Direct, dwOrigin = Internal **Note** There is no notification at P1:1111 after the call has been pickup.

OtherCallPickup

Configuration

Service parameter “Auto Call Pickup Enabled” is enabled.

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

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

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

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

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

P1:4000 and P1:4001 are configured.

Action	Expected Event
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)

Action	Expected Event
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:2000 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with OtherPickup option on P1:3000 **Note** Group DN is not required	Events on P1:3000: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 4001, Called = 1000, Connected = 4001, dwReason = Direct, dwOrigin = Internal

DirectCallPickup

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

Action	Expected Event
P1:4000 calls P1:1002 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with DirectCallPickup option with pickup groupDN (1000) on P1:10001	Events on P1:1001: LINE_NEWCALL and LINE_CALLSTATE with state = LINECALLSTATE_CONNECTED Call Info : Caller = 1001, Called = 1000, Connected = 4001, dwReason = Direct, dwOrigin = Internal

CallPickup (Negative Use Case)

Configuration

Service parameter Auto Call Pickup Enabled is enabled.

P1:2000 is already opened by the application.

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

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

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

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

Action	Expected events
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with CallPickup option on P1:2000	Line_Reply with Error LINEERR_OPERATIONUNAVAIL

GroupCallPickup with SuperSet Call PickupDN

Configuration

Service parameter Auto Call Pickup Enabled is enabled.

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

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

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

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

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

P1:4000 and P1:4001 are configured.

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

Action	Expected events
P1:4000 calls P1:2000 P1:4001 calls P1:1000	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111 Call 2: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupPickup option with pickup group(3333) on P1:3000	Line_Reply with Error LINEERR_CALLUNAVAIL

Group or Direct CallPickup with Invalid DN

Action	Expected events
LineIntialize OpenLines – 1000:P1	Line Open Successful
LineGetDevCaps with Extension Version – 000A0000 on P1:2000	CallPickUp Group DN and Partition Information will be sent to application.(P1:2222)
Application sends CciscoLineDevSpecificRegisterCallPickupGroupForNotification with DN and Partition info of PickUpGroup P1:1111	Line_Reply with success. LINE_CREATE event will sent to Application for P1:1111
LineOpen with new DeviceID	LineOpen Successful
P1:4000 calls P1:1002	Call1: LINE_APPNEWCALL LINE_CALLSTATE with State = LINECALLSTATE_UNKNOWN to Application on Line P1:1111
Application sends CciscoLineDevSpecificPickUpCallFromPickupGroup with GroupPickup option with pickup group(9999) on P1:3000 Variant -Direct Call Pickup with InValid DN	Line_Reply with Error LINEERR_OPERATIONFAILED Line_Reply with Error LINEERR_INVALLINESTATE

CCMEncryption Enhancements

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

Table 66: CiscoTSP Connecting to 10.x CUCM

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

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

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

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

Table 67: 9.x CiscoTSP Connecting to 10.x CUCM

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

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

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

Table 68: 9.x CiscoTSP Connecting to 10.x CUCM

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

CIUS Session Persistency

Notify the Line Application and Expose the Changed IP Address

Action	TAPI messages	TAPI structures
lineInitializeEx	lineDevices are Enumerated	
lineOpen for a lineDevice on the wireless device TAPI100	lineOpen() returns success	
lineGetDevCaps() with DeviceID = DeviceId of TAPI100	lineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.250" (FA1F4D0A -Little endian Hex format)
The device TAPI100 moves across WiFi networks resulting in change in the IPv4 address from 10.77.31.250 to 10.77.31.176 **Variation 1:** The device TAPI100 moves from a IPv4 n/w to a Ipv6 n/w with new ip as 2001:db8::1:0:0:1 **Variation 2:** The device TAPI100 is docked/undocked and hence changes from WAN/LAN to wireless network	EVENT = LINE_DEVSPECIFIC dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED dwParam2 = LPCT_DEVICE_IPADDRESS Variation result: 1) Same as above 2) Same as above	

Action	TAPI messages	TAPI structures
lineGetDevCaps() with DeviceID = DeviceId of TAPI100	lineGetDevCaps() returns success	LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176" (B01F4D0A -Little endian Hex format) Variation 1: LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv6_only RegisteredIPv6Address = "2001:db8::1:0:0:1" (Application should use the Offset and size fields of IPv6 address from LINEDEVCAPS to retrieve the value of IPv6 address) Variation 2: LINEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.176"

Notify the Phone Application and Expose the Changed IP Address

Action	TAPI Message	TAPI structures
phoneInitializeEx	phoneDevices are Enumerated	
phoneOpen for a phoneDevice of wireless device TAPI100	phoneOpen() returns success	
phoneGetDevCaps() with DeviceID = DeviceId of TAPI100	phoneGetDevCaps() returns success	PHONEDEVCAPS::DevSpecific RegisteredIPAddressMode = IPAddress_IPv4_only RegisteredIPv4Address = "10.77.31.250" (FA1F4D0A -Little endian Hex format)

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

Click to Conference

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

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

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

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

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

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

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

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

Drop Party by Using Click-2-Conference

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

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

Drop Entire Conference by Using Click-2-Conference Feature

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

Conference Enhancements

Noncontroller Adding Parties to Conferences

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

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

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

Chaining Two Ad Hoc Conferences Using Join

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

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

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

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

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

CTI Remote Device

Expose Remote Destination Info for CTI Remote Device in ProviderDeviceLineInfoEvent

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

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

Expose Remote Destination Info for CTI Remote Device in ProviderDeviceRegisteredWithLineInfoNotify

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

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

Expose New Device Type for CTI Remote Device

Precondition:

CTIRD (CTI Remote Device -Name: CTIRDdrajesh)

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

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

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

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

EP (Enter Prise Phone -SCCP -IP Phone)

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

CSF (CSF Device -Name: CSFdrajesh)

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

Remote Destination configured on CSF device:

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

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

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

Enumerating CTI Remote Devices and Exposing Remote Destination Information to Application

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

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

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

Precondition: In addition to above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

Precondition: continuation from previous UseCase

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

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

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

Remove RD From Admin -RD Info Change Notification to Application

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

Precondition: In addition to above usecase

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

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

Remote Destination Information on Non CTI RemoteDevice / CSF Device

Precondition: In addition to above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

Add RD From Application -RD Info Change Notification to Application

Precondition: Remove All RD's from Admin Page

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

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

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

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

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

Precondition: continuation from previous UseCase

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

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

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

Precondition: continuation from previous UseCase

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

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

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

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

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

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

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

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

Precondition: continuation from previous UseCase

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

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

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

Precondition: continuation from previous UseCase

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

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

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

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

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

Precondition: continuation from previous UseCase

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

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

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

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

Multiple Apps Setting Active RD

Precondition: same as UseCase 1

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

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

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

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

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

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

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

CTI/CCM Manager FailOver Scenario - Active RD

Precondition: same as UseCase 1

TSP is configured with Primary and Secondary CTI Manager

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

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

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

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

Precondition: same as UseCase 1

TSP is configured with Primary and Secondary CTI Manager

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

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

Monitoring CSF Device in Soft Phone/Desk Phone Mode

Precondition: continuation from previous UseCase

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

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

Precondition: continuation from previous UseCase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

Precondition: continuation from previous UseCase

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

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

Basic Incoming Call to CTI Remote Device

CTI remote device:

A (CTI Remote Device -Name: CTIRD1)

Remote Destination:

RD1 -Remote Destination configured on CTI Remote Device A

(Name: Mobile, Number: 914086271309)

RD2 -Remote Destination configured on CTI Remote Device A

(Name: Office, Number: 914089022131)

Line:

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

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

Enterprise Phones:

B (IP Phone -Name: SEPxxxxxxxx)

Line:

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

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

C (IP Phone -Name: SEPxxxxxxxx)

Line:

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

D (IP Phone -Name: SEPxxxxxxxx)

Line:

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

CSF Device:

D (CSF Device -Name: CSF-drajesh)

Remote Destination:

RD-01 -Remote Destination configured on CSF device D

(Name: CSF-Mobile, Number: 914086271309)

RD-02 -Remote Destination configured on CSF device D

(Name: CSF-Office, Number: 914089022131)

Line:

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

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

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

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

DVO Call (Outgoing Call Initiation From CTI Remote Device)

Precondition: same as above usecase

Action	TAPI messages	TAPI structures
LineInitializeEx	Lines are Enumerated	

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

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

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

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

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

Multiple Calls -Answer/Hold/Resume

Precondition: same as above usecase

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

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

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

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

Multiple Calls -Multiple Lines -Answer/Hold/Resume

Precondition: same as above usecase

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

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

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

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

Transfer

Precondition: same as above usecase

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

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

Direct Transfer on Same Line

Precondition: same as above usecase

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

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

Conference -Setupconference/AddtoConference

Precondition: same as above usecase

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

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

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

Join on Same Line

Precondition: same as above usecase

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

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

Direct Transfer/Join Across Line on CTI Remote Device

Precondition: same as above usecase

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

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

Cbarge

Precondition: same as above usecase

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

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

URI Dialing -Basic Incoming Call to CTI Remote Device

Precondition: InAddition to configuration from previous usecases

CTI Remote Device:

Line:

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

C (IP Phone -Name: SEPxxxxxxxx)

Line:

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

D (IP Phone -Name: SEPxxxxxxxx)

Line:

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

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

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

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

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

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

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

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

Precondition: same as above usecase

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

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

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

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

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

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

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

CTI RD Call Forwarding

Table 69: Use Case 1: Device A Calls CTIRD When Active RD Is Not Set and "Route calls to all remote destinations when client is not connected" Is Enabled.

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

Table 70: Use Case 2: Device A Calls CTIRD When Active RD Is Not Set and "Route calls to all remote destinations when client is not connected" Is Disabled. There Is No Call Forward Number Set on the Shared Enterprise Phone

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

Table 71: Use Case 3: Device A Calls CTIRD When CTI Remote Device Is Observed , Remote Destination Is Not Configured and "Route calls to all remote destinations when client is not connected" Is Enabled (CFNA Is Configured On Enterprise Number to Voice Mail Box)

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

Table 72: Use Case 4: Device A Calls CTIRD When CTI Remote Device Is Observed , Remote Destination Is Not Configured and "Route calls to all remote destinations when client is not connected" Is Disabled (CFNA Is Configured On Enterprise Number to Voice Mail Box)

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

Table 73: Use Case 5: DeviceA Calls CTIRD When Active RD Is Set and "Route calls to all remote destinations when client is not connected" Is Enabled. Setup: A IP Phone, B CTI-RD, C RDD1, D RDD2. Active RD Is Set to C

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

Table 74: Use Case 6: Device A Calls CTIRD When Active RD Is Set and "Route calls to all remote destinations when client is not connected" Is Enabled. Setup: A IP Phone, B CTI-RD, C RDD1, D RDD2. Active RD Is Set to C

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

Video Capabilities and Multimedia Information

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

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

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

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

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

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

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

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

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

Media Capability on Device A (ParkDN/Pickupdevice)

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

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

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

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

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

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

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

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

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

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

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

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

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

Action	Expected events
A does a Call to B, B answers. LineInitializeEx Issue LineGetcallInfo() with Ext version for linedevice B LineShutdown	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
Variation 1: A has video enabled and B has video disabled.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None
Variation 2: A has video enabled and B is a CTI Port or Route Point.	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None

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

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

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

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

Action	Expected events
Variation 1: A has video enabled and B has video disabled. A has Telepresence enabled and has 3 screens, B has Telepresence disabled and has 1 screens.	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapBitMask = 0x00000007 CalledPartyMultiMediaCapBitMask = 0x00000007 CallingPartyMultiMediaCapInfo : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000001(Telepresence Enabled) Screen Count = 3 CalledPartyMultiMediaCapInfo : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 1
Variation 2: A has video enabled,1 screen and B is a CTI Port or Route Point.	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapBitMask = 0x00000007 CalledPartyMultiMediaCapBitMask = 0x00000000 CallingPartyMultiMediaCapInfo : VideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 0x00000001 CalledPartyMultiMediaCapInfo : VideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_None] TelepresenceInfo = 0x000000000(Telepresence Disabled) Screen Count = 0x00000000

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

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

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

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

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

Action	Expected events
	RxIsMKIPresent = 0x00000000 SecurityIndicator = 0x00000001
Variation 1: A does a LineMakeCall to B / B answers the call Application does LineHold on B LineGetCallInfo on A and B	 CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000003
Application does LineUnHold on B LineGetCallInfo on A and B	CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000000
Application does a LineDrop on B. LineGetCallInfo on A and B	CallDevSpecific event returned -SLDSMT_MULTIMEDIA_STREAMSDATA The value of MediaMode should be changed 0x000000003

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

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. B answers. LineGetCallInfo on A LineGetCallInfo on B LineShutdown	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
Variation 1: A and B are SIP Phone and have video enabled. LineGetCallInfo on A LineGetCallInfo on B	A: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

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

Action	Expected events
	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
	C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
	D: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled]

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. LineGetCallInfo on B B redirects the call to C, LineGetCallInfo on C C redirects the call to D, LineGetCallInfo on D	

Action	Expected events
LineShutdown	

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

Action	Expected events

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

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

Shared Line – Hold and Resume Scenario Over SIP Trunk (Phone A and C Are Video-Enabled SIP Phones and Phone B Is Video-Disabled, Phone A Is in Cluster 1 and Phone B and C Are in Cluster 2. Phone B and C Are Shared Lines)

Action	Expected events

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

Action	Expected events
B Holds the call. C Unholds the call. LineGetCallInfo on A LineGetCallInfo on C	LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000000[CiscoDeviceVideoCapability_Disabled]

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

Action	Expected events

Action	Expected events
	B: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] D: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyMultiMediaCapabilityBitMask = 0x000000001 CalledPartyMultiMediaCapabilityBitMask = 0x000000001 CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]

Action	Expected events
LineInitializeEx A does a LineMakeCall to B. LineGetCallInfo on B B redirects the call to C. LineGetCallInfo on C	

Action	Expected events
C redirects the call to D. LineGetCallInfo on D LineShutdown	

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

Action	Expected events

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

Action	Expected events
LineGetCallInfo on A	C: LINEGETCALLINFO::DEVSPECIFIC exposes CallingPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled] CalledPartyVideoCapStatus = 0x000000001[CiscoDeviceVideoCapability_Enabled]
LineGetCallInfo on C	

Direct Transfer Across Lines

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

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

Direct Transfer Across Lines on RoundTable Phones via Application

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

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

Direct Transfer on Same Line on RoundTable Phones Via Application

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

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

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

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

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

Failure of Direct Transfer Calls Across Lines

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

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

Direct Transfer Calls Across Lines in Conference Scenario

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

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

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

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

Connect Transfer Across Lines on RoundTable Phones

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

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

Do Not Disturb-Reject

Application Enables DND-R on a Phone

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

Normal Feature Priority

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

Feature Priority - Emergency

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

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

Shared Line Scenario for DND-R

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

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

Application Disables DND-R or Changes the Option for DND

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

Drop Any Party

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

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

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

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

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

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

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

Shared Line-Scenario

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

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

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

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

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

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

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

Chained Conference

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

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

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

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

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

Early Offer

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

Application Dynamically Registers CTI Port with Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

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

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

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

Application Dynamically Registers CTI Port Without Early Offer Support

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Delayed Offer

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

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

Application Dynamically Registers IPV6 CTI Port with Early Offer Support

Configuration

A – CTI Port; CDC – IPV6 Only

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

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

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

Mutiple Applications Dynamically Register CTI Port/RP

Configuration

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

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

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

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

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

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

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

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

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

Action	TSP Message to application data
App1: Application sends lineDevSpecific(CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info	Line_Reply with Success
App2: Application sends LineDevSpecific (CciscoLineDevSpecificSetRTPParamsForCall) with IPAddress and Port Info different from the Info App1 has set.	Line_Reply with error LINEERR_OPERATIONUNAVAIL
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) LINE_DEVSPECIFIC dwParam1 = compressionType & SLDSMT_OPEN_LOGICAL_CHANNEL dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy (8 bits) – IPAddressingMode

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

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success

Action	TSP Message to application data
Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState dwParam1 = x040, InService
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster 2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)
*** Disconnect the Existing Call Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability -0x00000000 – to disable the Early Offer support	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster 2 Call is being routed through the SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING/ LINECALLSTATE_RINGBACK)
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED)

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

Configuration

A – CTI Port in Cluster1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info	Line_Reply with Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001 – to enable the Early Offer support	Line_Reply with Success
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode

Action	TSP Message to application data
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) Media will be set and Media Events will be Reported to Application
Variant 1: A – SCCP/SIP Phone	Behavior is same and new SLDSMT_RTP_GET_IP_PORT Notification will not be fired to application.

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

Configuration

A – CTI Port;

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

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

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

Configuration

A – CTI Port;

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Action	TSP Message to application data
During Installation of CiscoTSP User has to select New Wave Driver. lineInitialize	 Line_reply with Success Lines will be Enumerated to Application.
lineOpen() with Extversion – 0x000B0000 for Line A	Line_Open successful
LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success LineInserviceEvent reports to Application Line_LineDevState Dwparam1 = x040, InService
Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
Application sends lineDevSpecific(CCiscoLineDevSpecificSetStatusMsgs) with m_DevSpecificStatusMsgsFlag = DEVSPECIFIC_GET_IP_PORT -0x00000400	Line_Reply with Success
Application calls LineMakeCall() on A dialing a Party in Cluster2 Call is routed through SIP trunk with Early Offer Enabled	A: LINE_CALLSTATE (LINECALLSTATE_PROCEEDING) LINE_DEVSPECIFIC dwParam1 = SLDSMT_RTP_GET_IP_PORT dwParam2 = 0x00000xyy x (ninth Bit from LSB) – 0 – SetRTP (1-App has to set RTP / 0 – App need not set RTP) yy – IPAddressing Mode **Note** On this new Notification, applications has to Open the Port.

Action	TSP Message to application data
Other Party answers the Call	A: LINE_CALLSTATE (LINECALLSTATE_CONNECTED) Media will be set and Media Events will be reported to Application

Mutiple Applications Statically Register CTI Port

Configuration

A – CTI Port in Cluster 1

Cluster1 and Cluster2 connected via SIP trunk

SIP trunk Supports Early Offer

Applications:

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

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

Action	TSP Message to application data
App1 and App2: Both Connecting to same CTI Manager lineInitialize	Line_reply with Success Lines will be Enumerated to Application.
App1 and App2: lineOpen() with Extversion – 0x800B0000 for Line A	Line_Open successful
App1 and App2: LineSetStatusMessages() – with dwLinestates – 0xcc	LineSetStatusMessages returns Success
App1: Application sends lineDevSpecific(CciscoLineDevSpecificEnableFeatureSupport) with m_Feature – 0x00000001, m_Feature_Capability - 0x00000001	Line_Reply with Success
App1: Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to Register A	Line_Reply with Success LineInserviceEvent reports to Application.

Action	TSP Message to application data
App2: Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to Register A	Line_Devspecific fails with Error LINEERR_REGISTER_GETPORT_SUPPORT_MISMATCH
Variant: App1 and App2 connecting to different Cti Managers App2: (After App1 has already registered CtiPort -A) Application sends lineDevSpecific(CCiscoLineDevSpecificUserControlRTPStream) with MediaCaps Info to register CtiPort A	LineReply – success LINE_CLOSE for the CTI Port

End-To-End Call Trace

Direct Call Scenario: Variation 1

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

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

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

Direct Call Scenario: Variation 2

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

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

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

Consult Transfer Scenario: Variation 1

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

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

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

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

Consult Transfer Scenario: Variation 2

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

Action	CTI events	Expected Results
A calls B and B answers the call. B setups transfer to C and C answers the call	LineInitialize LineOpen on A , LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	
LineInitialize LineOpen on A, LineOpen on B, LineOpen on C with new ExtVesrion 0x000A0000	ExistingCallEvent received for A (Primary Call between A and B)	For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
	ExistingCallEvent received for B (Primary Call between A and B)	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For B

Action	CTI events	Expected Results
	ExistingCallEvent received for B (Consultation Call between B and C)	LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For C LINE_CALLDEVSPECIFIC event is received
	ExistingCallEvent received for C (Consultation Call between B and C)	dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A (Primary Call between A and B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B (Primary Call between A and B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on B (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B2
LineGetCallInfo on C (Consultation Call between B and C)		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1
Applications completes Transfer	CallGlobalCallHandleChangedEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1

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

Blind Transfer Scenario

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

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

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

Redirect Scenario

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

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

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

Shared Line Scenario

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

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

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

Shared Line Scenario with Barge

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

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

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

Action	CTI events	Expected results
	NewCallEvent received for B	
	NewCallEvent received for B’	
	CallGlobalCallHandleChangedEvent received for B	

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

Call Park Scenario: Variation 1

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

Service Parameter Preserve globalcallid For Parked Calls set to False

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

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

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

Call Park Scenario: Variation 2

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

Service Parameter Preserve globalcallid For Parked Calls set to True

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

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

3-Party Conference Call Scenario

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

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

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

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

Action	CTI events	Expected results
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

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

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

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

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

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

Conference Chaining Scenario Using Join

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

E drops from the conference.

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

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

Transfer Call Scenario via QSIP Without Path Replacement

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

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

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

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

Transfer Call Scenario via QSIP with Path Replacement

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

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

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

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

Hunt List Scenario

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

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

Action	CTI events	Expected results
LineInitialize LineOpen on A , LineOpen on B, LineOpen on C, LineOpen on D with new ExtVesrion 0x000A0000		
D calls Hunt Pilot DN.Call is first offered to Phone A, followed by B and then C.	NewCallEvent received for D NewCallEvent received for A	For D LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0 For A LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0

Action	CTI events	Expected results
	NewCallEvent received for B	For B LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
	NewCallEvent received for C	For C LINE_CALLDEVSPECIFIC event is received dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_UNIQUE_CALL_REF_ID_INFO dwParam3 = 0
LineGetCallInfo on D		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference D1
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on B		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference B1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C1

Call Pickup Scenario: Variation 1

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

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

Same Behaviour for Group Pickup.

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

Action	CTI events	Expected results
LineGetCallInfo on A		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference A1
LineGetCallInfo on C		LINECALLINFO::DEVSPECIFIC would contain Unique Call Reference C2

Call Pickup Scenario: Variation 2

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

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

Same Behaviour for Group Pickup.

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

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

EnergyWise Deep Sleep Mode Use Cases

Configuration

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

Verify EnergyWisePowerSavePlus Reason Code in LINEDEVSTATE Message

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

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	

Action	Expected result
Set Event filters for Inservice and Outofservice events. LinesetstatusMessage with dwlineStates flags LINEDEVSTATE_INSERVICE | LINEDEVSTATE_OUTOFSERVICE	CiscoTSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0 param3 = x0
When Phone A goes to Deep Sleep mode and unregisters	Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0
When PowerOntime is reached, Cisco Unified IP Phones Series 7900 device registers back to CUCM.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0

Variance

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

Verify EnergyWisePowerSavePlus Reason Code in PhoneState Suspend

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

Action	Expected result
PhoneInitialize	
PhoneOpen on A with ExtVersion xB0000 or higher	

Action	Expected result
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Phone A goes to Deep Sleep Mode and unregisters.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_ EnergyWisePowerSavePlus param3 = x0
When PowerOntime is reached, Cisco Unified IP Phones Series 7900 device registers back to CUCM.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0

Variance

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

Verify Reason EnergyWisePowerSavePlus Reason Code in LineDevstate/Phone State Message

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

Action	Expected result
LineInitialize	
LineOpen on A with ExtVersion xB0000 or higher	

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

Action	Expected result
For Cisco Unified IP Phones Series 9900 and 6900, press the Select Key to power up.	Cisco TSP Notifies LineInServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_INSERVICE param2 = x0, param3 = x0, Cisco TSP Notifies DeviceInServiceEvent to application through Phone state Event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_RESUME param2 = x0, param3 = x0,

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

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

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

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

Action	Expected result
PhoneOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Restart Call Manager services from serviceability page.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_CallManagerFailure param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_CallManagerFailure param3 = x0

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

Verify DeviceUnregister Reason Code in LineDevstate/Phone State Event

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

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

Action	Expected result
PhoneOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Manually unplug the Ethernet cable from device.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_DeviceUnregistered param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_DeviceUnregistered param3 = x0

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

Verify CTILinkFailure Reason Code in LineDevstate/Phone State Message

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

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

Action	Expected result
PhoneOpen on A with ExtVersion xB0000 or higher	
Set Event filters for Resume and Suspend events. For Example: PhonesetstatusMessage with dwPhoneStates flags PHONESTATE_SUSPEND | PHONESTATE_RESUME	
Stop CTI Manager services from serviceability page.	Cisco TSP Notifies DeviceOutOfServiceEvent to application through Phone state event. received PHONE_STATE device = xxx cbInst = x0 param1 = PHONESTATE_SUSPEND param2 = CiscoPhoneStateOutOfServiceReason_CTILinkFailure param3 = x0, Cisco TSP Notifies LineOutOfServiceEvent to application: received LINE_LINEDEVSTATE device = xxx cbInst = x0 param1 = LINEDEVSTATE_OUTOFSERVICE param2 = CiscoLineDevStateOutOfServiceReason_CTILinkFailure param3 = x0

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

Extension Mobility Cross Cluster

Common Configuration

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

TAPI Application Does LineInitializeEx and EMCC User Logs Into a Device

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

1. Open the TAPI Application with User A and do LineInitializeEx.
2. User A EM login to DeviceH on Cluster1.

TAPI Application Does LineInitializeEx and EMCCUser Logs Out of a Device

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

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

Application Does PhoneInitializeEx and EMCC User Logs In to a Device

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

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

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

Title	EMCC user logs out of a device
Description	Testing the scenario where TAPI Application does PhoneInitializeEx and EMCCUserLogs out of a Device
Test Setup	EM_Profile1 is included in application control list DeviceH is not in application control list

Expected Results	Step 2: Application receives PHONE _REMOVE for Line1

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

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

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

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

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

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

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

EMCC User Logs In to a Device with LineH Configured

Title	EMCC user logs in to a device with LineH configured
Description	Testing the scenario where EMCCUserLogin to a Device with LineH configured
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured

Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1

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

EMCC User Logs Out of a Device with LineH Configured

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

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

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

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

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

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

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

Description	Testing the scenario where EMCCUserLogin to a device with LineH configured and administrator removes the device from the Application Control list
Test Setup	EM_Profile1 is included in application control list DeviceH is included in application control list with LineH configured
Expected Results	Step 2: • Application receives LINE_REMOVE for LineH • Application receives LINE_CREATE for Line1 Step3: • Application will not receive any events.

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

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

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

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

4. Administrator removes the DeviceH from application control list.

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

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

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

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

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

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

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

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

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

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

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

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

Extension Mobility Memory Optimization Option

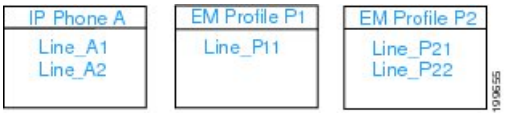
The following section describes common configuration and use cases for Early Offer Support.

Common Configuration

The message flow in the following figure is described below:

- IP Phone_A is configured in DB with lines Line_A1 and LineA2
- User1 has a device profile EM_Profile1 configured with Line_P11
- User2 has a device profile EM_Profile2 configured with lines Line_P21 and Line_P22

Figure 34: EM Memory Optimization Scenario 1



The message flow in the following figure is described below:

- Application uses Line_N to receive other-device state notifications

Figure 35: EM Memory Optimization Scenario 2

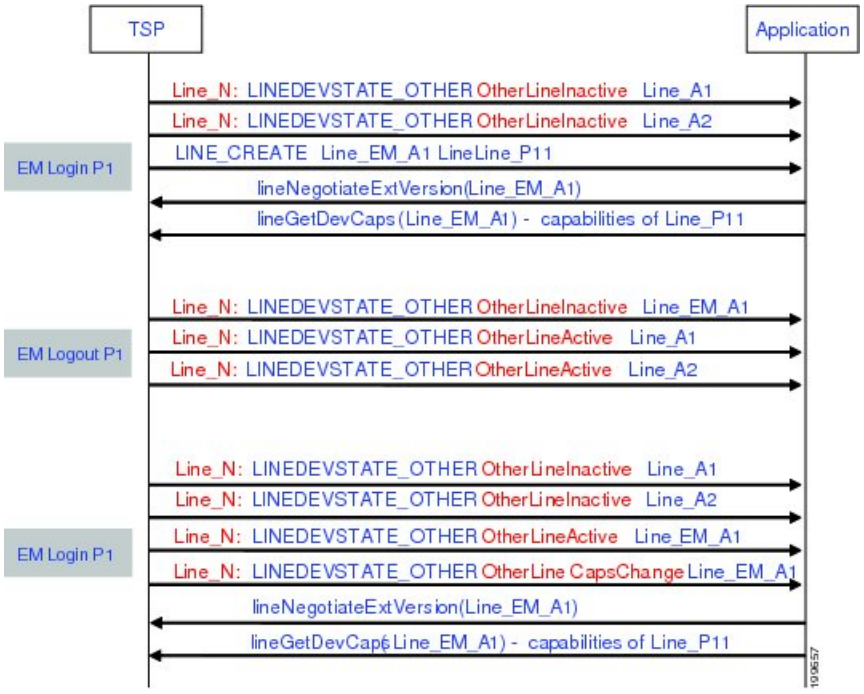
Use Cases

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

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

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

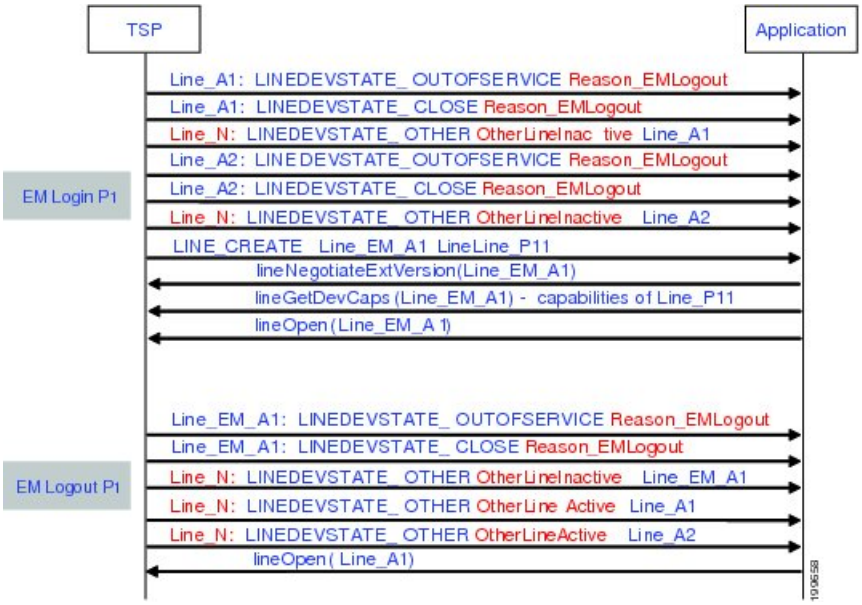
Figure 36: EM Memory Optimization Option Feature Use Case 1



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

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

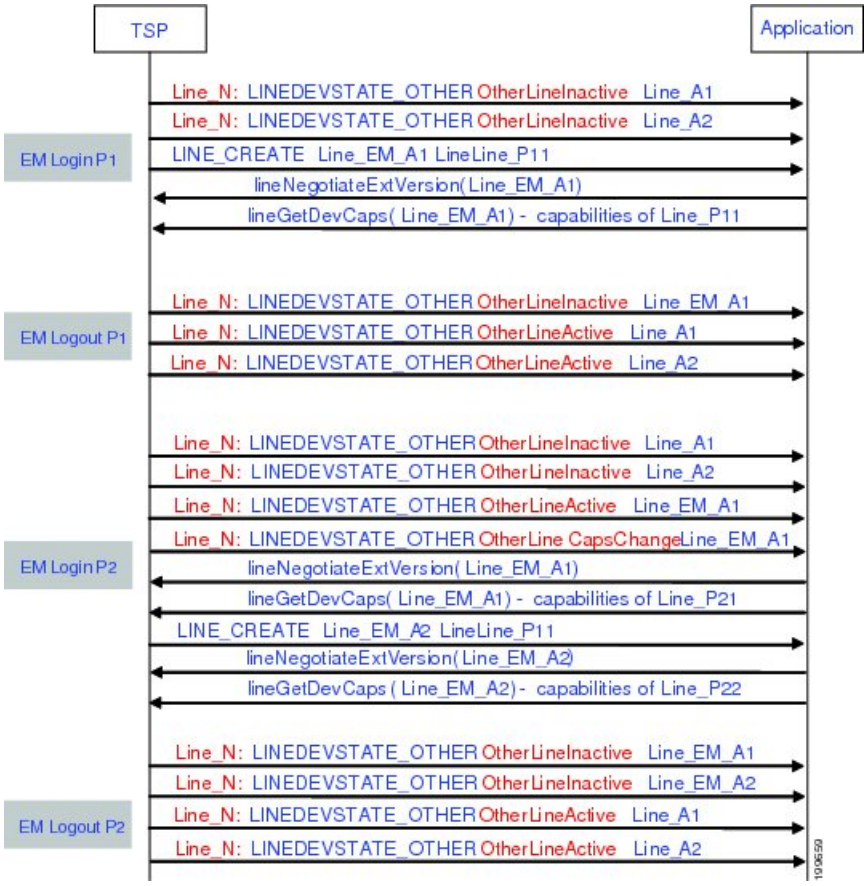
Figure 37: EM Memory Optimization Option Feature Use Case 2



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

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

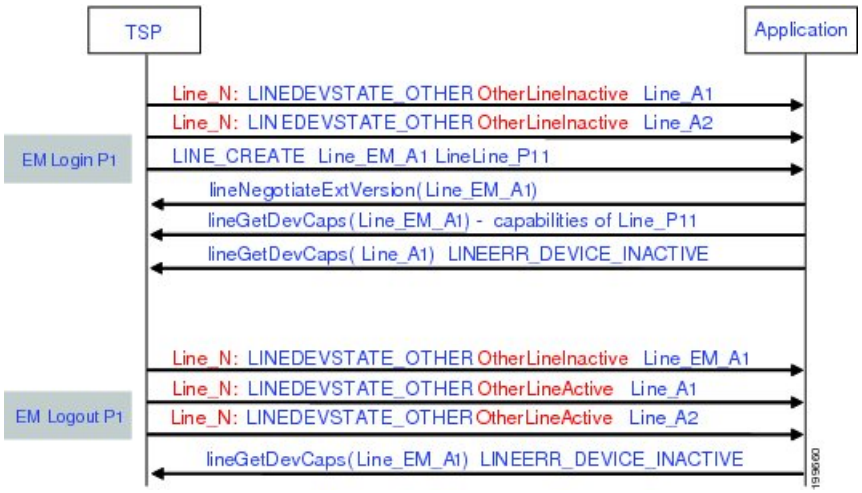
Figure 38: EM Memory Optimization Option Feature Use Case 3



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

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

Figure 39: EM Memory Optimization Option Feature Use Case 4



External Call Control

Basic Call Initiated From TAPI with External Call Control on Translation Pattern and CEPM Returns Reject

Configuration

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

Procedure

Application sends a lineMakeCall at A to call B.

Result

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

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

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

Basic Call Initiated From TAPI Using External Call Control on Translation Pattern and CEPM Returns Divert with Modified Calling and Called Parties

Configuration

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

Procedure

Application sends a lineMakeCall at A to call B.

Result

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

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

Call will be extended to C (modified calling and modified called in divert to routing directive, overrides the calling and called number transformation configured for translation pattern and the call is diverted to C)

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

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

Basic Call Initiated From TAPI Using External Call Control on Translation Pattern and CEPM Returns Continue with Modified Calling and Called Parties

Configuration

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

Procedure

Application sends a lineMakeCall at A to call B.

Result

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

CEPM returns continue with ModifiedCalling = MA and ModifiedCalled = MB

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

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

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

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

Configuration

Phone A, B, C are in cluster devices.

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

Procedure

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

Result

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

CEPM returns continue with ModifiedCalling = MB and ModifiedCalled = MC

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

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

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

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

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

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

Configuration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Party	TSP Message to App Data
C1 does a lineAddToConference	

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

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

Forced Authorization and Client Matter Code Scenarios

Manual Call to a Destination That Requires an FAC

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

Preconditions

Party A is Idle. Party B requires an FAC.

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

Table 75: Message Sequences for Manual Call to a Destination That Requires an FAC

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

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

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

Manual Call to a Destination That Requires Both FAC and CMC

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

Preconditions

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

Table 76: Message Sequences for Manual Call to a Destination That Requires Both FAC and CMC

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

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

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

lineMakeCall to a Destination That Requires an FAC

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

Preconditions

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

Table 77: Message Sequences for lineMakeCall to a Destination That Requires an FAC

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

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

lineMakeCall to a Destination That Requires Both FAC and CMC

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

Table 78: Message Sequences for lineMakeCall to a Destination That Requires Both FAC and CMC

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

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

Timeout Waiting for FAC or Invalid FAC

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

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

Table 79: Message Sequences for Timeout Waiting for FAC or Invalid FAC

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

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

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

Gateway Recording

Table 80: ClusterID and RecordType in LineGetDevCaps

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

Setup:

A is external caller.

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

Table 81: External Call to a CTI Remote Device Using Ingress Gateway for Forking with Selective Recording

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

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

Setup:

A is external caller.

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

Table 82: External Call to a CTI Remote Device Using Egress Gateway for Forking with Automatic Recording

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

Setup:

A is external caller.

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

Table 83: Initiate a Recording at CTIRD Follow by Hold and Resume the Call at the CTIRD

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

Setup:

A is external caller.

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

Table 84: Initiate a Recording at CTIRD Follow by Hold and Resume the Call at the Internal Other Party

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

Setup:

A, B are internal callers to the CTI RD

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

Table 85: Initiate a Recording at CTIRD Follow by Internal Other Party Redirects the Call to an Internal 3rd Party

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

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

Setup:

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

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

Table 86: Initiate a Recording at CTIRD Follow by External Other Party Redirects the Call to an External 3rd Party

Action	TSP Messages/Events
Application opens the provider.	

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

Setup:

A, B are internal callers to the CTI RD

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

Table 87: Initiate a Recording at CTIRD Follow by Internal Other Party Transfers the Call to an Internal 3rd Party

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

Setup:

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

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

Table 88: Initiate a Recording at CTIRD Follow by External Other Party Transfers the Call to an External 3rd Party

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

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

Setup:

A, B are internal callers to the CTI RD

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

Table 89: Initiate a Recording at CTIRD Follow by Internal Other Party Conferences an Internal 3rd Party

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

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

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

Setup:

A, B are internal callers to the CTI RD

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

Table 90: Initiate a Recording at CTIRD Follow by Restart Recording That Fails

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

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

Hunt List

Phones -A, B, C and X

Hunt Pilots: HP1

Member LG1, LG2, LG3

HP2.

Member LG11, LG12, LG13 are CTI port

Pickup Group1 : has LG1, lG2, LG3, X

Pickup Group2: has HP1, X

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

Basic Hunt List Call

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

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

Hunt List Call Moved to Next Member

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, Called Name = HP1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Events, requests and responses
LG2 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Hunt List Calls FWNA and FWNA Is Not Configured on HuntPilot

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

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

Hunt List Call FWNA with FWNA to B

Action	Events, requests and responses
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG2 to LG3	Call at LG2 goes IDLE At LG3: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

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

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

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

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

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

HuntList Call Redirected When in ACCEPT State

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

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

Hunt List Call Redirected When in Connected State

Table 91: Message Sequence for Hunt List Call Redirected When in Connected State

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

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

Hunt List Call Member Is CTI or RP Port

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

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

Table 92: Message Sequence for Hunt List Call Moved to Different Line Group Members and Answered by CTI Port

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

Hunt List Call Is Redirected to Another Hunt List

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

Action	Events, requests and responses
A redirects the call to HP2 and call offered to LG11	At A: Call goes IDLE At LG1: LINE_CALLSTATE -RINGBACK Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A At LG11: LINE_CALLSTATE -ACCEPTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 RedirectionID = HP2 RedirectingID = A

Action	Events, requests and responses
LG11 answers the call	At LG1: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectingID = A RedirectionID = HP2 At LG11: LINE_CALLSTATE -OFFERING Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A

Hunt List Call Is Consult Transferred to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup transfer to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = LG1

Action	Events, requests and responses
LG1 completes transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = LG1 At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = LG1

Hunt List Call Direct Transferred to Another Line

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1
LG1 calls to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 performs Direct Transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = LG1 At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = LG1

Hunt List Call Is Conferenced to Another Line

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

Action	Events, requests and responses
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 completes conference	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B

Action	Events, requests and responses
	Called = A Connected = A

Hunt List Call Is Joined to Another Line

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

Action	Events, requests and responses
LG1 calls B, B answers the call	At LG1 Call-1: ONHOLD Caller = A Called = HP1 HuntPilot = HP1 Connected = A Call-2: CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 performs Join	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B

Action	Events, requests and responses
	Called = A Connected = A

Hunt List Call Is Conferenced to Another Hunt List After LG11 Answers

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

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP2 ->LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2

Action	Events, requests and responses
	At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1 HP name = -empty CONFERECED Caller = LG11 Called = A Connected = A

Hunt List Call Conferenced to the Same Hunt List and Completes Conference Before Hunt List Agent Answers

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

Action	Events, requests and responses
LG1 setup conference to HG1, where alerting on LG2,	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A RINGBACK Caller = LG1 Called = HP1 HuntPilot = HP1 At LG2: LINE_CALLSTATE -ACCEPTED Caller = LG1 Called = HP2 HuntPilot = HP2

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = HP1 HuntPilot = HP1 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = HP1 HuntPilot = HP1

Action	Events, requests and responses
	At LG2: ACCEPTED CONFERECED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HG1 CONFERECED Caller = LG2 Called = A Connected = A

Action	Events, requests and responses
LG2 answers the call	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 Called Name = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 ConnectedName = LG2 HuntPilot = HP1 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 Connected = A CONFERENCED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 Called = A Connected = A

Action	Events, requests and responses
	At LG2: CONNECTED CONFERECED Caller = LG1 Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HG1 CONFERECED Caller = LG11

Hunt List Basic Call with SharedLine

LG1’ is sharedline with LG1

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

Hunt List Basic Call with DND-R Configured on LG1

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG2 since LG1 has DND enabled.. Then LG2 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Hunt List Call Put in Conference via Join Operation

Action	Events, requests and responses
B calls A, A answer	At A: Call-1 LINE_CALLSTATE -CONNECTED Caller = B Called = A Connected = B At G: LINE_CALLSTATE -CONNECTED Caller = B Called = A Connected = A

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: Call-1 is on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Events, requests and responses
Application initiates JOIN calls on A with final call as call-1	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = B Called = A Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = B Called = A Connected = A CONFERECED Caller = B Called = LG1 HuntPilot = HP1

Action	Events, requests and responses
	Connected = LG1 HuntPilot = HP1

Hunt List Call Is Picked Up From Pickup Group -G-Pickup Auto Pick Pp Is Enabled

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
Line X got notification of the call	Got call pickup notification of call offering at LG1

Action	Events, requests and responses
Line X does group pick from LG1	At A: LINE_CALLSTATE -CONNECTED Caller = X Called = HP1, HuntPilot = HP1 ConnectedID = X At X: LINE_CALLSTATE -PROCEEDING Caller = X Called = PickGroup# LINE_CALLSTATE -CONNECTED Caller = X Called = PickGroup#, ConnectedID = A

Hunt List Call Is Picked Up From Pickup Group When LG1 Is in Pickup Group 1 -Auto Pickup Disabled

Action	Events, requests and responses
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1
Line X got notification of the call	Got call pickup notification of call offering at LG1

Action	Events, requests and responses
Line X does group pick from LG1	Original pickup call goes IDLE
X got server call about the pickup call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1, HuntPilot = HP1 ConnectedID = X At X: new call offered at X from server, and answer LINE_CALLSTATE -CONNECTED Caller = A Called = X ConnectedID = A

Hunt List Call Is Picked Up From Pickup Group When HP2 Is in Pickup Group 2 -Auto Pick Up Enabled

Action	Events, requests and responses
App initiates call from A to HP2 and the call is offered at LG11	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP2, HuntPilot = HP2 At LG11: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP2, HuntPilot = HP2
Line X got notification of the call	Got call pickup notification of call offering at HP2

Action	Events, requests and responses
Line X does group pick from HP2	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP2, HuntPilot = HP2 ConnectedID = X At X: LINE_CALLSTATE -CONNECTED Caller = X Called = PickGroup#, ConnectedID = A

Conferenced Hunt List Call Becomes Two-Party Call

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

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 Called Name = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2

Action	Events, requests and responses
	At LG11: CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 CONFERECED Caller = LG11 Called = A Connected = A
LG11 drops call	At A: Conf Parent call goes IDLE CONFERENCED call to LG11 goes IDLE CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: Conf Parent call goes IDLE CONFERENCED call to LG11 goes IDLE CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A At LG11: Calls go IDLE

Hunt List Broadcast Scenario (Broadcast Option Is Configured on HP1)

Action	Events, requests and responses
App initiates call from A to HP1, and call is offered at LG1, LG2 and LG3	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 At LG2: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 At LG3: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1

Note HP Broadcast is not supported when interacting with Call PickUp feature.

Hunt List Call Is Involved in c-Barge Conference

LG1’ is sharedline with LG1

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

Action	Events, requests and responses
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = B Connected = B At LG1': LINE_CALLSTATE -CONNECTED INACTIVE Caller = A Called = HP1 HuntPilot = HP1 Connected = A LINE_CALLSTATE -CONNECTED INACTIVE CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Called Name = B Connected = B Called Name = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B

Action	Events, requests and responses
	At LG1': CONNECTED INACTIVE CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Action	Events, requests and responses
LG1' cBarges in	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B CONFERENCED Caller = A Called = LG1' Connected = LG1' At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1 Called = B Connected = B CONFERENCED Caller = LG1 Called = LG1' Connected = LG1'

Action	Events, requests and responses

Action	Events, requests and responses
	CONNECTED INACTIVE CONFERECED Caller = LG1’ Called = LG1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1’ Called = B Connected = B CONFERENCED Caller = LG1’ Called = A Connected = A At LG1’: CONNECTED CONFERECED Caller = LG1’ Called = LG1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = LG1’ Called = B Connected = B CONFERENCED Caller = LG1’ Called = A

Action	Events, requests and responses
	Connected = A CONNECTED INACTIVE
	CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B CONFERENCED Caller = LG1 Called = LG1’ Connected = LG1’ At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A CONFERENCED Caller = B Called = LG1’ Connected = LG1’

Hunt List Feature Interact with Four-Party Conference

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

Action	Events, requests and responses
LG1 setup conference to HG2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HG2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HG1

Action	Events, requests and responses
LG1 completes conference	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = HG2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2

Action	Events, requests and responses
	Connected = LG11 HuntPilot = HG2
	CONFERECED Caller = LG11 Called = A Connected = A
LG1 setup conference to X, X answers the call	At LG1: ONHOLDPENDINGCONFERENCE CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 CONNECTED Caller = LG1 Called = X Connected = X At X: CONNECTED Caller = LG1 Called = X Connected = LG1

Action	Events, requests and responses
LG1 completes conference	

Action	Events, requests and responses
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 CONFERENCED Caller = A Called = X Connected = X At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERECED Caller = LG1 Called = HP2 Connected = LG11 HuntPilot = HP2 CONFERENCED

Action	Events, requests and responses
	Caller = LG1 Called = X Connected = X
	At LG11: CONNECTED CONFERECED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HG1 CONFERECED Caller = LG11 Called = A Connected = A CONFERENCED Caller = LG11 Called = X Connected = X

Hunt Pilot Connected Number Feature

HP1 and HP2 are 2 Huntpilots with configuration "Display Line Group Member DN as Connected Party" set.

HP1: LG1, LG2, LG3(LineGroup/MemberDNs

HP2: LG4, LG5, LG6(LineGroups/MemberDNs

Table 93: Basic Hunt List Call

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

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = A ModifiedRedirectingID = ModifiedRedirectionID =

Table 94: Hunt List Call Moved to Next Member

Action	Expected events
App initiates call from A to HP1 and call is offered to LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1
Call moves from LG1 to LG2	Call at LG1 goes IDLE At LG2: LINE_CALLSTATE -ACCEPTED Caller = A, Called = HP1, HuntPilot = HP1

Action	Expected events
LG2 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG2 HuntPilot = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG2 At LG2: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = A
Variance : perform the test with all HuntList algorithm Top-Down algorithm Circular algorithm Longest Idle Time algorithm	

Table 95: Hunt List Call Is Redirected When It Is in Connected State

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1	At A: LINE_CALLSTATE -RINGBACK Caller = A Called = HP1, HuntPilot = HP1 At LG1: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN:ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN :ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID =

Action	Expected events
LG1 answers the call	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CPN:ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID = At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CPN :ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = LG1 ModifiedRedirectingID = ModifiedRedirectionID =

Action	Expected events
LG1 redirects call to B	At A : LINE_CALLSTATE -RINGBACK Caller = A Called = HP1 HuntPilot = HP1 Connected = RedirectingID = HP1 RedirectionID = B CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = ModifiedRedirectingID = [LG1] ModifiedRedirectionID = B At LG1: Call goes IDLE At B: LINE_CALLSTATE -ACCEPTED Caller = A Called = HP1, HuntPilot = HP1 RedirectingID = HP1 RedirectionID = B CPN: ModifiedCalling = A ModifiedCalled = [LG1] Modifiedconnected = ModifiedRedirectingID = LG1 ModifiedRedirectionID = B

Table 96: Hunt List Call -member Is CTI / RP Port

Action	Expected events
Same as ,Table 93: Basic Hunt List Call, on page 867 but with CTI port	Similar expectation as of Basic Hunt Call.

Table 97: Hunt List Call Moved to Different Line Group Members and Answered by CTI Port

Action	Expected events
Same as ,Table 94: Hunt List Call Moved to Next Member, on page 869 but with CTI port	Similar expectation as of Hunt List call moved to next member.

Table 98: Hunt List Call Is Redirected to Another Hunt List

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Expected events
A redirects the call to HP2 and call offered to LG11	At A: Call goes IDLE At LG1: LINE_CALLSTATE -RINGBACK Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A CPN: ModifiedCalling = LG1 ModifiedCalled = HP1 Modifiedconnected = ModifiedRedirectingID = A ModifiedRedirectionID = HP2 At LG11: LINE_CALLSTATE -ACCEPTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 RedirectionID = HP2 RedirectingID = A CPN:ModifiedCalling = LG1 ModifiedCalled = HP2 Modifiedconnected = ModifiedRedirectingID = A ModifiedRedirectionID = HP2

Action	Expected events
LG11 answers the call	At LG1: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectingID = A RedirectionID = HP2 CPN: ModifiedCalling = LG1 ModifiedCalled = HP1 Modifiedconnected = LG11 ModifiedRedirectingID = A ModifiedRedirectionID = LG11 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP2, HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = HP2 RedirectingID = A CPN: ModifiedCalling = LG1 ModifiedCalled = HP2 Modifiedconnected = LG1 ModifiedRedirectingID = A ModifiedRedirectionID = LG11

Table 99: Hunt List Call Is Consult Transferred to Another Line

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup transfer to B, B answer	At LG1 Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = LG1

Action	Expected events
LG1 completes transfer	At A : LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = B RedirectionID = B RedirectingID = HP1 CPN: ModifiedCalling = A ModifiedCalled = HP1 Modifiedconnected = B ModifiedRedirectingID = LG1 ModifiedRedirectionID = B At LG1: both call goes IDLE At B: LINE_CALLSTATE -CONNECTED Caller = A Called = B Connected = A RedirectionID = B RedirectingID = HP1 CPN: ModifiedCalling = A ModifiedCalled = B Modifiedconnected = A ModifiedRedirectingID = LG1 ModifiedRedirectionID = B

Table 100: Hunt List Call Is Conferenced to Another Line

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A
LG1 setup conference to B, B answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONNECTED Caller = LG1 Called = B Connected = B At B: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = B Connected = B

Action	Expected events
LG1 completes conference	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = B Connected = B At LG1: CONNECTED CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED Caller = LG1 Called = B Connected = B At B: CONNECTED CONFERENCED Caller = LG1 Called = B Connected = LG1 CONFERECED Caller = B Called = A Connected = A

Table 101: Hunt List Call Is Conferenced to Another Hunt List After LG11 Answers

Action	Expected events
App initiates call from A to HP1 and the call is offered at LG1 , and LG1 answers	At A: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 At LG1: LINE_CALLSTATE -CONNECTED Caller = A Called = HP1 HuntPilot = HP1 Connected = A

Action	Expected events
LG1 setup conference to HP2, where alerting on LG11, LG11 answers the call	At LG1 ONHOLDPENDINGCONF CONFERECED Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HP1

Action	Expected events
LG1 completes conference	

Action	Expected events
	At A: CONNECTED CONFERENCED Caller = A Called = HP1 HuntPilot = HP1 Connected = LG1 HuntPilot = HP1 CONFERENCED Caller = A Called = LG11 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG1: CONNECTED CONFERECED [A-LG1] Caller = A Called = HP1 HuntPilot = HP1 Connected = A CONFERENCED[LG1-LG11] Caller = LG1 Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: CONNECTED CONFERECED [LG11-LG1] Caller = LG1 Called = HP2 HuntPilot = HP2

Action	Expected events
	Connected = LG1
	CONFERECED [LG11-A] Caller = LG11 Called = A Connected = A

Caller Consult Transfer Call to Another Hunt List

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

Action	Events, requests and responses
A setup transfer to HP2, offered at LG11, LG11 anwser	At A Call-1 is put on HOLD Call-2 LINE_CALLSTATE -CONNECTED Caller = A Called = HP2 HuntPilot = HP2 Connected = LG11 HuntPilot = HP2 At LG11: LINE_CALLSTATE -CONNECTED Caller = A Called = HP2 HuntPilot = HP2 Connected = A

Action	Events, requests and responses
A completes transfer	At LG1 : LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP1 HuntPilot = HP1 Connected = LG11 HuntPilot = HP2 RedirectionID = LG11 RedirectingID = A At A: both call goes IDLE At LG11: LINE_CALLSTATE -CONNECTED Caller = LG1 HuntPilot = HP1 Called = HP2 HuntPilot = HP2 Connected = LG1 HuntPilot = HP1 RedirectionID = LG11 RedirectingID = A

Intercom

This configuration gets used for all the following use cases:

1. IPPhone A has two lines, line1 (1000) and line2 (5000). Line2 represents an intercom line. Speeddial to 5001 with label ìAssistant_1î gets configured.
2. IPPhone B has three lines, line1 (1001), line2 (5001), and Line3 (5002). Line2 and Line3 represent intercom lines. Speeddial to 5000 with label ìManager_1î gets configured on line2. Line 3 does not have Speeddial configured for it.
3. IPPhone C has two lines, line1 (1002) and line2 (5003). 5003 represents an intercom line that is configured with Speeddial to 5002 with label ìAssistant_5002î.

4. IPPhone D has one line (5004). 5004 represnts an intercom line.
5. CTIPort X has two lines, line1 (2000) and line2 (5555). Line2 represents an intercom line. Speedial to 5001 gets configured with label ìAssistant_1î.
6. Intercom lines (5000 to 5003) exists in same partition = Intercom_Group_1 and they remain reachable from each other. 5004 exists in Intercom_Group_2.
7. Application monitoring all lines on all devices.

Assumption: Application initialized and CTI provided the details on speeddial and lines with intercom line on all the devices. Behavior should act the same for phones that are running SCCP, and those that are running SIP.

Application Invoking Speeddial

Action	Events
LineOpen on 5000 & 5001 Initiate InterCom Call on 5000	For 5000 receive LINE_CALLSTATE cbInst = x0 param1 = x03000000 param2 = x1, ACTIVE param3 = x0, Receive StartTransmission event For 5001 receive LINE_CALLSTATE cbInst = x0 param1 = x03000000 param2 = x1, ACTIVE param3 = x0, Receive StartReception event Receive zipzip tone with reason as intercom

Agent Invokes Talkback

Action	Events
Continuing from the previous use case, 5001 initiates LineTalkBack from application on the InterCom call	For 5000 receive LINE_CALLSTATE device = x10218 param1 = x100, CONNECTED param2 = x1, ACTIVE param3 = x0, Receive StartReception event For 5001 receive LINE_CALLSTATE device = x101f6 cbInst = x0 param1 = x100, CONNECTED param2 = x1, ACTIVE param3 = x0, Receive StartTransmission event

Change the SpeedDial

Action	Events
Open line 5000 LineChangeSpeeddial request (speeddial to 5003, label = “Assistant_5003”)	The new speed dial and label is successfully set for the intercom line Receive LineSpeeddialChangeEvent from CTI Send LINE_DEVSPECIFIC to indicate that speeddial and label changed
Application issues LIneGetDevCaps to retrieve speeddial/label that is set on the line	TAPI returns configured speeddial/label that is configured on the line.

IPv6 Use Cases

The use cases related to IPv6 are provided below:

Register CTI Port with IPv4 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv4

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv4 only on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv4. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is able to register CTI Port with IPv4 address.

Register CTI Port with IPv6 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv6

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv6 only on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv6. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is not able to register CTI Port. TSP returns error LINEERR_OPERATIONUNAVAIL

Register CTI Port with IPv6 When Unified CM Is IPv6 Disabled and Common Device Configuration Is IPv4_v6

Steps	Expected result
1. Enterprise parameter for IPv6 is disabled. IP addressing mode for CTI Port = IPv4_v6 on common device config page. 2. Open provider and do a LineNegotiateExtensionVersion with the higher bit set on both dwExtLowVersion and dwExtHighVersion 3. Application does a LineOpen with new Ext ver. The lineopen will be delayed till user specifies the Addressing mode 4. Application uses CCiscoLineDevSpecificSetIPAddressMode to set the addressing mode as IPv6. Application uses CciscoLineDevSpecificSendLineOpen to trigger Lineopen.	Application is not able to register CTI Port. TSP returns error LINEERR_OPERATIONUNAVAIL

IPv6 Phone A Calls IPv6 Phone B

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. 2. Open two lines A and B 3. Phone A which is IPv6 calls Phone B which is IPv6 4. Events at Phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: Blank FarEndIPAddressIpv6: IPv6 address of A
1. While Media is established: • Events on phone A	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of B. ReceptionRTPDestinationAddress = IPv6 address of A.
• Event on phone B	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of A. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv4_v6 Phone Calls IPv6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. **2.** Open two lines A and B **3.** Phone A which is IPv4_v6 calls Phone B which is IPv6 **4.** Events at Phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: IPv4 address of A FarEndIPAddressIpv6: IPv6 address of A
1. While Media is established: • Events on phone A	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of B. ReceptionRTPDestinationAddress = IPv6 address of A.
• Event on phone B	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of A. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv4 Phone Calls IPv6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. **2.** Open two lines A and B **3.** Phone A which is IPv4 calls Phone B which is IPv6 **4.** Events at Phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: IPv4 address of A FarEndIPAddressIpv6:
1. While Media is established: • Events on phone A	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv4 address of MTP Resource. ReceptionRTPDestinationAddress = IPv4 address of A.
• Event on phone B	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of B.

IPv6 Phone Calls IPv4 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. **2.** Open two lines A and B **3.** Phone A which is IPv6 only calls Phone B which is IPv4 **4.** Events at Phone B	FireCallState = Offering, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: FarEndIPAddressIpv6: IPv6 address of A
1. While Media is established: • Events on phone A	Do a GetLinecallInfo, LineCallInfo will contain the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of A.
• Event on phone B	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv4 address of MTP Resource. ReceptionRTPDestinationAddress = IPv4 address of B.

IPv6 Phone Calls IPv4_v6 Phone

Steps	Expected result
1. Enterprise parameter for IPv6 is enabled. **2.** Phone A which is IPv6 only calls Phone B which is IPv4_v6 only. **3.** Open lines A and B	
4. Events at Phone B	Existing Call, Do a GetlineCallInfo. LineCallInfo contains the following in devspecific part, FarEndIPAddress: FarEndIPAddressIpv6: IPv6 address of A
1. While Media is established: • Events on phone A	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of MTP Resource. ReceptionRTPDestinationAddress = IPv6 address of A.
• Event on phone B	Do a GetLinecallInfo, LineCallInfo contains the following in devspecific part, TransmissionRTPDestinationAddress = IPv6 address of Phone A. ReceptionRTPDestinationAddress = IPv6 address of B.

Common Device Configuration Device Mode Changes From IPv4_v6 to IPv4

Steps	Expected result
User changes the device configuration on common device configuration from IPv4_v6 to IPv4 only	Application receives LineDevSpecific for the opened CTI Ports/RP in the device config indicating that Addressing mode has changed. All lines registered as IPv6 get a LINE_CLOSE Event. Application can then re-register these lines later.

Common Device Configuration Device Mode Changes From IPv4 to IPv6

Steps	Expected result
User changes the device configuration on common device configuration from IPv4 only to IPv6 only	Application receives LineDevSpecific for the opened CTI Ports/RP in the device config indicating that Addressing mode has changed. All lines registered as IPv4 get a LINE_CLOSE Event. Application can then re-register these lines later.

Join Across Lines

Setup

Line A on device A

Line B1 and B2 on device B

Line C on device C

Line D on device D

Line B1' on device B1', B1' is a shared line with B1

Join Two Calls From Different Lines to B1

Action	Expected events
A ‡ B1 is HOLD	For A
C ‡ B2 is connected	LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1: LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For B1': LINE_CALLSTATE param1 = x100, CONNECTED, INACTIVE Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C

Action	Expected events
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)
	For B1’
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C

Join Three Calls From Different Lines to B1

Action	Expected events
A ‡ B1 is hold,	
C ‡ B2 is hold	
D ‡ B2 is connected	For A:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2:
	LINE_CALLSTATE for call-1

Action	Expected events
	param1 = x100, HOLD Caller = C, Called = B2 , Connected = C
	LINE_CALLSTATE for call-2
	param1 = x100, CONNECTED Caller = D, Called = B2 , Connected = D
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For D:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = D, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	CONFERENCED Caller = A Called = D, Connected = D
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D
	For B2
	Call-1 and call-2 will go IDLE
	For C

Action	Expected events
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = C Called = A, Connected = A
	CONFERENCED Caller = C Called = D, Connected = D
	For D
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = D Called = A, Connected = A
	CONFERENCED Caller = D Called = C, Connected = C
	For B1’
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D

Join Calls From Different Lines to B1 with Conference

Action	Expected events
A,B1,C in conference where B1 is controller	For A:
D‡ B2 Connected	
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = A Called = C, Connected = C
	For B1:
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For B2:
	LINE_CALLSTATE for call-1

Action	Expected events
	param1 = x100, CONNECTED Caller = D, Called = B2 , Connected = D
	For C:
	CONNECTED
	CONFERENCED Caller = C, Called = A, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	For D:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = D, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	CONFERENCED Caller = A Called = D, Connected = D
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D
	For B2
	Call will go IDLE
	For C

Action	Expected events
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = C Called = A, Connected = A
	CONFERENCED Caller = C Called = D, Connected = D
	For D
	CONNECTED
	CONFERENCED Caller = B1, Called = C, Connected = B1
	CONFERENCED Caller = D Called = A, Connected = A
	CONFERENCED Caller = D Called = C, Connected = C
	For B1’
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C
	CONFERENCED Caller = B1 Called = D, Connected = D

Join Two Calls From Different Lines to B1 While B1 Is Not Monitored by TAPI

Action	Expected events
A ‡ B1 is HOLD,	
C ‡ B2 is connected	For A:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:
	LINE_CALLSTATE

Action	Expected events
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
User issues join request from phone with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)

Join Two Calls From Different Lines to B2

Action	Expected events
A ‡ B1 is HOLD,	
C ‡ B2 is connected	For A:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:

Action	Expected events
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For B1':
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B1 as survival call	For A
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	CONFERENCED Caller = A Called = C, Connected = C
	For B1
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C ??
	For B2
	Call will go IDLE
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B1 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B1)
	For B1'
	CONNECTED INACTIVE
	CONFERENCED Caller = A, Called = B1, Connected = A
	CONFERENCED Caller = B1 Called = C, Connected = C

Action	Expected events
A ‡ B1 is HOLD,	For A:

Action	Expected events
B1 issues setup conference	
C ‡ B2 is connected	
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = A, Called = B1 Connected B1
	For B1:
	Primary call
	LINE_CALLSTATE
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B1
	Consult call
	DIALTONE
	For B2:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2 , Connected = C
	For C:
	LINE_CALLSTATE
	param1 = x100, CONNECTED Caller = C, Called = B2, Connected = B2
	For B1’:
	LINE_CALLSTATE
	param1 = x100, HOLD Caller = A, Called = B1, Connected = A
Application issues lineDevSpecific(SLDST_JOIN) with the call on B2 as survival call	For A:
	CONNECTED
	CONFERENCED Caller = A, Called = B1, Connected = B2
	CONFERENCED Caller = A Called = C, Connected = C
	For B1

Action	Expected events
	Both calls will go IDLE
	For B2
	CONNECTED
	CONFERENCED Caller = B1, Called = A, Connected = A
	CONFERENCED Caller = C Called = B1, Connected = C
	For C
	CONNECTED
	CONFERENCED Caller = C, Called = B2, Connected = B2 (or A)
	CONFERENCED Caller = C Called = A, Connected = A (or B2)
	For B1’
	Calls go IDLE

B1 Performs a Join Across Line Where B1 Is Already in a Conference Created by A

Action	Expected events
A, B1, C are in a conference created by A	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For A:

Action	Expected events
	B2 calls D, D answers
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For B2:
	Connected -Caller = B2, Called = D, Connected = D
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Connected -Caller = B2, Called = D, Connected = B2
B1 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = D, Connected = D
	For B1:
	Conference – Caller = A, Called = B1, Connected = B1
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = D, Connected = D
	For B2:

Action	Expected events
	Call will go IDLE
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = D, Connected = D
	For D:
	Conference – Caller = B1, Called = D, Connected = B1
	Connected
	Conference – Caller = D, Called = A, Connected = A
	Conference – Caller = D, Called = C, Connected = C

B2 Performs a Join Across Line Where B1 Is Already in a Conference Created by A

Action	Expected events
A,B1,C are in a conference created by A	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
B2 calls D, D answers	For A:
	Conference – Caller = A, Called = B1, Connected = B1

Action	Expected events
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For B2:
	Connected -Caller = B2, Called = D, Connected = D
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For D:
	Connected -Caller = B2, Called = D, Connected = B2
B2 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B2
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = D, Connected = D
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = D, Connected = D
	For B2:
	Call will go IDLE
	For C:

Action	Expected events
	Conference – Caller = B2, Called = C, Connected = B2
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = D, Connected = D
	For D:
	Conference – Caller = B2, Called = D, Connected = B2
	Connected
	Conference – Caller = D, Called = A, Connected = A
	Conference – Caller = D, Called = C, Connected = C

B1 Performs a Join Across Line Where B1 Is in One Conference and B2 Is in a Separate Conference

Action	Expected events
A,B1,C are in conference1	For A (GCID-1):
D, B2, E are in conference2	
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	For B1 (GCID-1):
	Conference – Caller = A, Called = B1, Connected = A
	OnHold
	Conference – Caller = B1, Called = C, Connected = C
	For C (GCID-1):
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	For D (GCID-2):
	Conference – Caller = D, Called = B2, Connected = B2
	Connected

Action	Expected events
	Conference – Caller = D, Called = E, Connected = E
	For B2 (GCID-2):
	Conference – Caller = D, Called = B2, Connected = D
	Connected
	Conference – Caller = B2, Called = E, Connected = E
	For E (GCID-2):
	Conference – Caller = B2, Called = E, Connected = B2
	Connected
	Conference – Caller = E, Called = D, Connected = D
B1 issues a lineDevSpecific(SLDST_JOIN) to join the calls on B1 and B2.	For A:
	Conference – Caller = A, Called = B1, Connected = B1
	Connected
	Conference – Caller = A, Called = C, Connected = C
	Conference – Caller = A, Called = CFB-2, Connected = CFB-2
	For B1:
	Conference – Caller = A, Called = B1, Connected = A
	Connected
	Conference – Caller = B1, Called = C, Connected = C
	Conference – Caller = B1, Called = CFB-2, Connected = CFB-2
	For B2:
	Call will go IDLE
	For C:
	Conference – Caller = B1, Called = C, Connected = B1
	Connected
	Conference – Caller = C, Called = A, Connected = A
	Conference – Caller = C, Called = CFB-2, Connected = CFB-2
	For D:

Action	Expected events
	Connected
	Conference – Caller = D, Called = E, Connected = E
	conference – Caller = D, Called = CFB-1, Connected = CFB-1
	For E:
	Connected
	Conference – Caller = E, Called = D, Connected = D
	Conference – Caller = E, Called = CFB-1, Connected = CFB-1

Logical Partitioning

Use cases related to Logical Partitioning feature are mentioned below:

Basic Call Scenario

Basic Call scenario ; Logical partitioning Enabled = true	
Description	Basic Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) on one Geolocation A calls B: LineMakeCall on A Dails B (DN) Variant 1: B Geo-Location was not Configured;B(PSTN);Policy Config : Interior to Interior Variant 2: B (PSTN) on another GeoLocation
Expected Results	Variant 1: Call will be successful; Reason: LP_IGNORE. Variant 2: A goes to Proceeding State and then On A there will be a DISCONNECTED call state will be sent to application with cause as LINEDISCONNECTMODE_UNKNOWN.

Redirect Scenario

Redirect scenario ; Logical partitioning Enabled = true	
Description	Redirect Call failure due to Logical partitioning Feature Policy.

Redirect scenario ; Logical partitioning Enabled = true	
Test Setup	Two Clusters (Cluster1 and Cluster2) configured with logical partition policy that will restrict the VOIP calls from Cluster1 to PSTN calls on Cluster2. (vice versa PSTN to VIOP) A on Cluster1 (VOIP) B on Cluster2 (VOIP) C on Cluster2 (PSTN) A calls B B redirects the call to C
Expected Results	Operation fails with error code LINEERR_OPERATION_FAIL_PARTITIONING_POLICY. Error code is processed on Cluster2
Variants	For Forward Operation same behaviour will be observed.

Transfer Call Scenario

Transfer Call scenario ; Logical partitioning Enabled = true	
Description	Transfer Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) in one GeoLocation (GeoLoc 1) B (VOIP) in another GeoLocation(GeoLoc 2) C (PSTN)in same GeoLocation as B (GeoLoc 2) A calls B SetUpTransfer on B. On Consult Call at B; Dials C. Complete Transfer on B.
Expected Results	Operation fails with error code "LINEERR_OPERATIONUNAVAIL".
Variants	For Operation Adhoc Conference same behaviour will be observed.

Join Scenario

Join scenario; Logical partitioning Enabled = true	
Description	Join failure due to Logical partitioning Feature Policy.

Join scenario; Logical partitioning Enabled = true	
Test Setup	A (VOIP) in one GeoLocation (GeoLoc 1) B (VOIP) in another GeoLocation(GeoLoc 2) C (VOIP)in same GeoLocation as B (GeoLoc 2) D (PSTN) in same GeoLocation as B (GeoLoc 2) B has Three Calls 1. B -> A 2. B -> C 3. B -> D Variant 1: Join on B with B -> A as Primary Call. Variant 2: Join on B with B -> D as Primary Call. Variant 3: Join on B with B -> C as Primary Call.
Expected Results	Variant 1: A, B and C will be in conference. Variant 2: B, C and D will be in conference. Variant 3:Either A or D will be in conference with B and C.

Shared Line Scenario

CallPickUp scenario ; Logical partitioning Enabled = true	
Description	CallPickUp Failure due to Logical partitioning Feature Policy.
Test Setup	A (PSTN) on one Geolocation -GeoLoc1 B (VOIP) on one Geolocation -GeoLoc1 C (VOIP) on one Geolocation -GeoLoc2 A Dails B B Parks the call C does LineUnPark
Expected Results	Call will be successful on A and A' call will not be present
Variants	Shared line features like barge, cbarge, hold & remote resume should be disabled for calls.

CallPark: Retrieve Scenario

CallPickUp scenario ; Logical partitioning Enabled = true	
Description	CallPickUp Failure due to Logical partitioning Feature Policy.

CallPickUp scenario ; Logical partitioning Enabled = true	
Test Setup	A (PSTN) on one Geolocation -GeoLoc1 B (VOIP) on one Geolocation -GeoLoc1 C (VOIP) on one Geolocation -GeoLoc2 A Dails B B Parks the call C does LineUnPark
Expected Results	CallUpark Will fail with error code "LINEERR_OPERATIONUNAVAIL".

Basic Call Scenario

Basic Call scenario ; Logical partitioning Enabled = true	
Description	Basic Call failure due to Logical partitioning Feature Policy.
Test Setup	A (VOIP) on one Geolocation A calls B: LineMakeCall on A Dails B (DN) Variant 1: B Geo-Location was not Configured;B(PSTN);Policy Config: Interior to Interior Variant 2: B (PSTN) on another GeoLocation
Expected Results	Variant 1: Call will be successful; Reason: LP_IGNORE. Variant 2: A goes to Proceeding State and then On A there will be a DISCONNECTED call state will be sent to application with cause as LINEDISCONNECTMODE_UNKNOWN.

Manual Outbound Call

The following table describes the message sequences for Manual Outbound Call when party A is idle.

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

Action	CTI messages	TAPI messages	TAPI structures
3. Party B accepts call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change
4. Party B answers call	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStartReceptionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartReception dwParam2 = IP Address dwParam3 = Port	No change
	CallStartTransmissionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartTransmission dwParam2 = IP Address dwParam3 = Port	No change

Note LINE_DEVSPECIFIC events are sent only if the application has requested them by using lineDevSpecific().

Monitoring and Recording

Monitoring a Call

A (agent) and B (customer) get connected. BIB on A gets set to on.

Action	CTI messages	TAPI messages	TAPI structures
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
C(supervisor) issues start monitoring req with A's permanentLineID as input	NewCallEvent, CH = C3, GCH = G2, Calling = C, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = ORIGIN dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = REASON, CALLERID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectingID = NP
A's BIB automatically answers	Party C		
	CallStateChangedEvent, CH = C3, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = C, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = A dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP
	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	MonitoringStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_MONITOR_STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	Party C		
	LineCallAttributeInfoEvent, CH = C3, Type = 2 (MonitorCall_Target), CI = C1, Address = A's DN, Partition = A's Partition, DeviceName = A's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = C dwCalledID = A dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: CallAttribute_SilentMonitorCall_Target, CI = C1, DN = A's DN, Partition = A's Partition, DeviceName = A's Name
	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	LineCallAttributeInfoEvent, CH = C1, Type = 1 (MonitorCall), CI = C3 Address = C's DN, Partition = C's Partition, DeviceName = C's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: CallAttribute_SilentMonitorCall, CI = C3 DN = C's DN, Partition = C's Partition, DeviceName = C's Name
C drops the call	Party C		
	CallStateChangedEvent, CH = C3, State = Idle, Cause = CauseNoError, Reason = Direct, Calling = C, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	
	Party A		
	MonitoringEndedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_MONITOR_ENDED dwParam2 = DisconnectMode_Normal dwParam3 = 0	

Automatic Recording

Recording type on A (agent phone) is configured as Automatic. D is configured as a Recorder Device.

Action	CTI messages	TAPI messages	TAPI structures
A recieves a call from B, and A answers the call Recording session gets established between the agent phone and the recorder	Party A		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = B, Called = A, OrigCalled = A, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	RecordingStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_ STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP

Action	CTI messages	TAPI messages	TAPI structures
	LineCallAttributeInfoEvent CH = C1, Type = 3 (Automatic Recording), Address = D's DN, Partition = D's Partition, DeviceName = D's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: App Controlled Recording, DN = D's DN, Partition = D's Partition, DeviceName = D's Name

Application-Controlled Recording

A (C1) and B (C2) connect. Recording Type on A gets configured as 'Application Based'. D gets configured as a Recorder Device.

Action	CTI messages	TAPI messages	TAPI structures
A issues start recording request Recording session gets established between the agent phone and the recorder	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	RecordingStartedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_STARTED dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP
	LineCallAttributeInfoEvent CH = C1, Type = 4 (App Controlled Recording), Address = D's DN, Partition = D's Partition, DeviceName = D's Name	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_ATTRIBUTE_INFO dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP DevSpecifc Data: Type: App Controlled Recording, DN = D's DN, Partition = D's Partition, DeviceName = D's Name

Action	CTI messages	TAPI messages	TAPI structures
A issues stop monitoring request	RecordingEndedEvent, CH = C1	LINE_CALLDEVSPECIFIC hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = SLDSMT_RECORDING_ENDED dwParam2 = DisconnectMode_Normal dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = A dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP

NuRD (Number Matching for Remote Destination) Support

Park Monitoring

Use cases related to Park Monitoring feature are mentioned below:

Park Monitoring Feature Disabled

Setup:

The Park Monitoring message flag is disabled by default.

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001)

All lines are monitered by TSP

Action	Expected events
1. A(3000) calls B(3001) **2.** B(3001) receives the call and parks the call	Application will not be notified about the New Parked call through LINE_NEWCALL event as the park Monitoring flag is disabled.

Park Monitoring Feature Enabled

Setup:

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001),C(3002)

All lines are monitered by TSP

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001).	Park Status Event on B: At Step 3: Application will be notified about the New Parked call through LINE_NEWCALL event At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. Application does a LineGetCallInfo.
1. A(3000) calls B(3001) **2.** B(3001) receives the call and parks the call at 5555	LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 2 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 2: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call at 5555 **4.** The Park Monitoring Reversion Timer expires while the call is still parked.	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 3 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 3: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No Retrieve destination configured on B(3001) as C(3002) **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked.	Park Status Event on B: At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded Application will receive the LINE_CALLSTATE event with callstate IDLE. The reason code CtiReasonforwardedNoRetrieve will be updated in the LINECALLINFO::dwDevSpecificData.ExtendedCallInfo. dwExtendedCallReason = CtiReasonforwardedNoRetrieve.
1. The Park Monitoring Forward No Retrieve timer expires and now the call is forwarded to the Park Monitoring Forward No Retrieve Destination C(3002).	Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 4: 1. The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). 2. A(3000) calls B(3001) 3. B(3001) receives the call and parks the call 4. A(3000) hangs up the call.	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Abandoned. Application will receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo.
	LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName TransactionIDID = Sub1. dwBearerMode: ParkStatus = 4 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 5: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call **4.** The Park Monitoring Reversion Timer Expires while the call is still parked. **5.** C(3002) retrieves the call	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Retrieved. Application will receive the LINE_CALLSTATE event with callstate IDLE.
	Application does a LineGetCallInfo. hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 5 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled: ParkedParty = 3000 dwCalledIDName: ParkedPartyPartition = P1.

Action	Expected events
Scenario 6: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination not configured. **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked **6.** The Park Monitoring Forward No Retrieve timer expires and the call is forwarded to the Parkers line.	Park Status Event on B At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded. Application will receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo.
	LineCallInfo will contain the following: hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled: ParkedParty = 3000 dwCalledIDName: ParkedPartyPartition = P1.

Action	Expected events
Scenario 7: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination configured as self(Parkers Line) **3.** A(3000) calls B(3001) **4.** B(3001) receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked **6.** The Park Monitoring Reversion Timer Expires while the call is still parked **7.** The Park Monitoring Forward No Retrieve timer expires and the call is forwarded to the Parkers line.	Park Status Event on B At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 6: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder. At Step 7: Application will receive the LINE_CALLSTATE event with the Park Status = Forwarded. Application will receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline: LH = 1 dwCallID: CallID dwReason: LINECALLREASON_PARKED dwRedirectingIDName: TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID: ParkDN = 5555 dwCallerName: ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Parked Call Exists

Setup:

Cisco Unified IP phones (future version) running SIP: A(3000), B(3001).

B is not monitered by TSP.

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call **4.** Now the Line B(3001) is monitered by TSP	Park Status Event on B: At Step 4: Application will be notified about the Parked call through LINE_NEWCALL event.when ever cisco TSP recives the LINE_PARK_STATUS event for already parked call. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName TransactionIDID = Sub1. dwBearerMode: ParkStatus = 2 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Shared Line Scenario

Setup:

A(3000) ,D(3003) are Cisco Unified IP phones (future version) running SIP

B(3001) and B'(3001) are shared lines for Cisco Unified IP phones (future version) running SIP

C(3002) and C'(3002) are shared lines where C is a Cisco Unified IP phone (future version) running SIP and C' is a Cisco Unified IP Phone 7900 Series running SIP .

For the shared lines the events will be delivered to the phone which parks the call .Events will not be delivered to the other phone though the line is shared.

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) and B'(3001) starts ringing. B(3001) receives the call and parks the call **4.** Park Monitoring reversion timer expires while the call is still parked. **5.** D(3003) retrieves the call	Park Status Event on B: At Step 3: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Reminder.
	At Step 5: Application will receive the LINE_CALLSTATE event with the Park Status = Retrieved Application will receive the LINE_CALLSTATE event with callstate IDLE.
	Application does a LineGetCallInfo. hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName :TransactionIDID = Sub1. dwBearerMode: ParkStatus = 5 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Action	Expected events
Scenario 2: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** The Park Monitoring Forward No retrieve destination configured as B(3001) **3.** A(3000) calls B(3001) **4.** B(3001) and B'(3001) starts ringing. B(3001)receives the call and parks the call **5.** The Park Monitoring Reversion Timer Expires while the call is still parked. **6.** The Park Monitoring Forward No Retrieve timer expires and call is forwarded to B(3001).Both B(3001) and B'(3001) starts ringing as they are shared lines.	Park Status Event will be sent only to B not B'. At Step 4: Application will receive the LINE_CALLSTATE event with the Park Status = Parked. At Step 5: Application receives the LINE_CALLSTATE event with the Park Status = Reminder. At Step 6: Application receives the LINE_CALLSTATE event with the Park Status = Forwarded.
	Application receive the LINE_CALLSTATE event with callstate IDLE. Application does a LineGetCallInfo. LineCallInfo contains the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName : TransactionIDID = Sub1. dwBearerMode: ParkStatus = 6 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.
Scenario 3: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls C(3002) **3.** C(3002) and C'(3002) starts ringing. C'(3002) receives the call and parks the call **4.** D(3003) retrieves the call	Park Status Event on C'. At Step 3: Application is notified about the New Parked call through LINE_NEWCALL event as the call is parked by the Normal TNP phone.

Park Monitoring Feature Disabled

Setup:

The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for line B(3001).

A(3000), D(3003) is a Cisco Unified IP phones (future version)

Application invokes the Line_open () API on provider to monitor ParkDN

.

Action	Expected events
Scenario 1: **1.** The Park Monitoring message flag is Enabled using SLDST_SET_STATUS_MESSAGES request for Line B(3001). **2.** A(3000) calls B(3001) **3.** B(3001) receives the call and parks the call **4.** The Park Monitoring Reversion Timer Expires while the call is still parked.	Park Status Event on B: At Step 3: Application receives the LINE_NEW_CALL event for PARKDN. At Step 3: Application receives the LINE_PARK_STATUS event with the Park Status = Parked. At Step 4: Application will receive the LINE_CALL_STATE event with the Park Status = Reminder. Application does a LineGetCallInfo. LineCallInfo will contain the following: hline : LH = 1 dwCallID : CallID dwReason :LINECALLREASON_PARKED dwRedirectingIDName :TransactionIDID = Sub1. dwBearerMode: ParkStatus = 3 dwCallerID : ParkDN = 5555 dwCallerName : ParkDNPartition = P1 dwcalled : ParkedParty = 3000 dwCalledIDName : ParkedPartyPartition = P1.

Persistent Connection Use Cases

The following pre-conditions apply to all persistent call use cases, unless specified:

- The provider is in IN_SERVICE state.
- All addresses and terminals are already in service.
- Device A (CTI Remote Device - Name: "CTIRDtapi", Line A1 (dn: 881000))

Remote destination 1 (Name: "rd", Number: "78000")

- Device B (IP Phone - Name: "SEP001319ACCA26", Line B1 (dn: 1000))
- Device C (IP Phone - Name: "SEP00156247EE60", Line C1 (dn: 2000))
- User1 has in its control list: Devices A, B and C. All devices and lines are observed.

Table 102: Call createPersistentCall() on an Address That Is Not Configured to a Remote Terminal Device, i.e. on an IP Phone

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall ("SEP00156247EE60", "5000", "remote") on device C.	Caught exception com.cisco.jtapi.PlatformException: Internal callprocessing error :Device does not support the command	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. COMMAND_NOT_IMPLEMENTED_ ON_DEVICE.

Table 103: Call createPersistentCall()on an Address That Is Configured to a Remote Terminal Device Where Active RD Is Not Set

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	Caught exception com.cisco.jtapi.PlatformException: The active remote destination is not set.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_REMOTE_DEVICE_REQUEST_ FAILED_ ACTIVE_RD_NOT_SET.

Table 104: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD Is Set. Verify That Persistent Call Is Connected

Action	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination ("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.

Action	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
User1 invokes CiscoAddress. getPersistentConnection ("CTIRDjtapi") and verify that the connection for the persistent call is returned and uses that to get the Call object and confirm it is for the persistent call.		((CiscoAddress. getPersistentConnection("CTIRDjtapi")). getCall()).isPersistentCall() = true.
User1 invokes Provider.getCalls()		Provider.getCalls() = null
User1 invokes Address.getConnections() on line A.		Address.getConnections() on line A = null
User1 invokes Terminal.getTerminal Connections() on device A.		Terminal.getTerminalConnections() on device A = null

Action	Events	Call Info
Disconnect/drop the persistent call. User1 invokes either Call.drop() or Connection.disconnect()	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 105: Call createPersistentCall() on an Address Configured to a Remote Terminal Device Where a Persistent Call Already Exists

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "6000", "remote2") on device A.	Caught exception com.cisco.jtapi.PlatformException: Persistent Call exists.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_PERSISTENT_CALL_EXISTS.

Table 106: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD Is Set. Verify That Persistent Call Is Connected and Then Have Remote Destination Hang Up

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", true) on device A.	CiscoProvTerminalRemote DestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.

Actions	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Remote destination with dn = 78000 hangs up.	GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 107: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True. Verify That Persistent Call Is Connected. Set Active RD = False and Verify That Persistent Call Is Dropped

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Actions	Events	Call Info
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", true) on device A	CiscoProvTerminal RemoteDestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", false) on device A.	CiscoProvTerminal RemoteDestinationChangedEv See persistent call gets dropped: GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = false

Table 108: Call createPersistentCall() on an Address That Is Configured to a Remote Terminal Device and Where Active RD = True. Verify That Persistent Call Is Connected. Make Incoming Customer Call to Same Remote Terminal Device

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", true) on device A.	CiscoProvTerminal RemoteDestinationChangedEv	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = true.

Actions	Events	Call Info
User1 invokes CiscoAddress. createPersistentCall ("CTIRDjtapi", "5000", "remote") on device A.	GC1: CallActiveEv GC1: ConnCreatedEv 8881000 GC1: ConnInProgressEv 8881000 GC1: CallCtlConnOfferedEv 8881000 GC1: ConnCreatedEv 5000 GC1: ConnConnectedEv 5000 GC1: CallCtlConnEstablishedEv 5000 GC1: ConnAlertingEv 8881000 GC1: CallCtlConnAlertingEv 8881000 GC1: TermConnCreatedEv CTIRDjtapi GC1: TermConnRingingEv CTIRDjtapi GC1: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000
Call answered at remote destination, dn = 78000	GC1: ConnConnectedEv 8881000 GC1: CallCtlConnEstablishedEv 8881000 GC1: TermConnActiveEv CTIRDjtapi GC1: CallCtlTermConnTalkingEv CTIRDjtapi	CallingAddress = 5000, CalledAddress = 8881000, CurrentCallingAddress = 5000, CurrentCalledAddress = 8881000

Actions	Events	Call Info
Call.connect("SEP001319ACCA26", "1000", "8881000")	GC2: CallActiveEv GC2: ConnCreatedEv 1000 GC2: ConnConnectedEv 1000 GC2: CallCtlConnInitiatedEv 1000 GC2: TermConnCreatedEv SEP001319ACCA26 GC2: TermConnActiveEv SEP001319ACCA26 GC2: CallCtlTermConnTalkingEv SEP001319ACCA26 GC2: CallCtlConnDialingEv 1000 GC2: CallCtlConnEstablishedEv 1000 GC2: ConnCreatedEv 8881000 GC2: ConnInProgressEv 8881000 GC2: CallCtlConnOfferedEv 8881000 GC2: ConnAlertingEv 8881000 GC2: CallCtlConnAlertingEv 8881000 GC2: TermConnCreatedEv CTIRDjtapi GC2: TermConnRingingEv CTIRDjtapi GC2: CallCtlTermConnRingingEv CTIRDjtapi	CallingAddress = 1000, CalledAddress = 8881000, CurrentCallingAddress = 1000, CurrentCalledAddress = 8881000
Call is answered at device A	GC2: ConnConnectedEv 8881000 GC2: CallCtlConnEstablishedEv 8881000 GC2: TermConnActiveEv CTIRDjtapi GC2: CallCtlTermConnTalkingEv CTIRDjtapi	
User1 invokes CiscoRemoteTerminal. setActiveRemoteDestination("78000", false) on device A.	CiscoProvTerminal RemoteDestinationChangedEv Both persistent call with GC1 and customer call with GC2 are not dropped/disconnected even though active rd = false.	A.getActiveRemoteDestinations() = CiscoRemoteDestinationInfo[1]. CiscoRemoteDestinationInfo[0]. getRemoteDestinationNumber() = "78000" CiscoRemoteDestinationInfo[0]. getIsActiveRD() = false.

Actions	Events	Call Info
Customer call with GC2 is disconnected/dropped. User1 invokes either Call.drop() or Connection.disconnect() on the call with GC2.	GC2: TermConnDroppedEv SEP001319ACCA26 GC2: CallCtlTermConnDroppedEv SEP001319ACCA26 GC2: ConnDisconnectedEv 1000 GC2: CallCtlConnDisconnectedEv 1000 GC2: TermConnDroppedEv CTIRDjtapi GC2: CallCtlTermConnDroppedEv CTIRDjtapi GC2: ConnDisconnectedEv 8881000 GC2: CallCtlConnDisconnectedEv 8881000 GC2: CallInvalidEv Since there are no active calls on device A and active rd is now false, the persistent call with GC1 is now dropped/disconnected. GC1: ConnDisconnectedEv 5000 GC1: CallCtlConnDisconnectedEv 5000 GC1: TermConnDroppedEv CTIRDjtapi GC1: CallCtlTermConnDroppedEv CTIRDjtapi GC1: ConnDisconnectedEv 8881000 GC1: CallCtlConnDisconnectedEv 8881000 GC1: CallInvalidEv	

Table 109: Have a Persistent Call and Customer Call Connected. Invoke hold() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		

Actions	Events	Call Info
Invoke hold() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 110: Have a Persistent Call and Customer Call Connected. Invoke startRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke startRecording() on the persistent call with GC1.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 111: Have a Persistent Call and Customer Call Connected. Invoke stopRecording() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke stopRecording() on the persistent call with GC1. Make sure Selective call recording is enabled.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 112: Have a Persistent Call and Customer Call Connected. Invoke conference() on the Persistent Call Where Persistent Call Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	

Actions	Events	Call Info
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where persistent call with GC1 is the primary call and customer call with GC2 is the secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 113: Have a Persistent Call and Customer Call Connected. Invoke conference() on the Persistent Call Where Persistent Call Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke conference() where customer call with GC2 is primary call and persistent call with GC1 is secondary call (jtapi internally calling join() for this).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 114: Have a Persistent Call and Customer Call Connected. Invoke park() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke park().	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 115: Have a Persistent Call and Customer Call Connected. Invoke transfer() on the Persistent Call Where Pc Is Primary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(Call) where persistent call with GC1 is primary call and customer call with GC2 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 116: Have a Persistent Call and Customer Call Connected. Invoke transfer() on the Persistent Call Where Pc Is Primary to Another Dn Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(String address) where persistent call with GC1 is primary call to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 117: Have a Persistent Call and Customer Call Connected. Invoke transfer() on the Persistent Call Where Pc Is Secondary Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke transfer(Call) where customer call with GC2 is primary call and persistent call with GC1 is secondary.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 118: Have a Persistent Call and Customer Call Connected. Invoke consult() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Make consult call from device A to line C (dn = 2000).	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 119: Have a Persistent Call and Customer Call Connected. Invoke pickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke pickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 120: Have a Persistent Call and Customer Call Connected. Invoke otherPickup() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke otherPickup("8881000") on device A.	Caught exception com.cisco.jtapi.PlatformException: Operation not allowed.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Table 121: Have a Persistent Call and Customer Call Connected. Invoke redirect() on the Persistent Call Which Should Be Rejected

Actions	Events	Call Info
User1 opens Provider and adds a provider observer.	ProvInServiceEv	
Assume already have a persistent call with GC1 and customer call with GC2.		
Invoke redirect("2000") on the persistent call.	Caught exception com.cisco.jtapi.PlatformException: Operation is not allowed on a Persistent Call.	Let "ex" be an instance of PlatformException: ((CiscoJtapiException) ex).getErrorCode() = CiscoJtapiException. CTIERR_OPERATION_NOT_ALLOWED_ ON_PERSISTENT _CALL.

Presentation Indication

Making a Call Through Translation Pattern

The following table describes the message sequences for the Presentation Indication scenario of making a call through translation pattern. In the Translation Pattern admin pages, both the callerID/Name and ConnectedID/Name get set to "Restricted".

Action	CTI messages	TAPI messages	TAPI structures
Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change
Party A dials Party B through Translation pattern	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
Party B accepts the call	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPartyPI = Allowed, Called = B, CalledPartyPI = Restricted, OrigCalled = B, OrigCalledPI = restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCallerIDName = A's Name dwCalledID = B dwCalledIDName = B’s name dwConnectedID = NP dwConnectedIDName = NP dwRedirectionID = NP dwRedirectionIDName = NP dwRedirectionID = NP dwRedirectionIDName = NP

Action	CTI messages	TAPI messages	TAPI structures
Party B accepts the call (continued)	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPI = Allowed, Called = B, CalledPI = Restricted, OrigCalled = B, OrigCalledPI = Restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwRedirectionID = NP dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP
Party B answers the call	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, CallingPI = Allowed, Called = B, CalledPI = Restricted, OrigCalled = B, OrigCalledPI = Restricted, LR = NP	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCallerIDName = A's Name dwCalledID = B dwCalledIDName = B’s Name dwConnectedID = A, dwConnectedIDName = A's Name, dwRedirectingID = NP dwRedirectingIDName = NP dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStartReceptionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartReception dwParam2 = IP Address dwParam3 = Port	No change
	CallStartTransmissionEvent, DH = A, CH = C1	LINE_DEVSPECIFIC1 hDevice = hCall-1 dwCallBackInstance = 0 dwParam1 = StartTransmission dwParam2 = IP Address dwParam3 = Port	No change

Note LINE_DEVSPECIFIC events only get sent if the application requested them by using lineDevSpecific().

Blind Transfer Through Translation Pattern

The following table describes the message sequences for the Presentation Indication scenario of Blind Transfer through Translation Pattern. In this scenario, A calls via translation pattern B, B answers, and A and B are connected.

Action	CTI messages	TAPI messages	TAPI structures
Party B does a lineBlindTranfser() to blind transfer call from party A to party C via translation pattern	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CallingPartyPI = Restricted, CalledChanged = True, Called = C, CalledPartyPI = Restricted, OriginalCalled = NULL, OriginalCalledPI = Restricted, LR = NULL, Cause = BlindTransfer	LINE_CALLINFO, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = CONNECTEDID, REDIRECTINGID, REDIRECTIONID	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B's name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C2, State = Idle, Reason = Direct, Calling = A, CallingPartyPI = Restricted, Called = B, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = NULL	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B's name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party B does a lineBlindTranfser() to blind transfer call from party A to party C via translation pattern (continued)	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	NewCallEvent, CH = C3, origin = Internal_Inbound, Reason = BlindTransfer, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_APPNEWCALL hDevice = C dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = TRANSFER dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = NP dwCalledIDName = NP dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party C is offering	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangeEvent, CH = C1, State = Ringback, Reason = Direct, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = B dwCalledIDName = B's name dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP
Party C is offering (continued)	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C3, State = Offering, Reason = BlindTransfer, Calling = A, CallingPartyPI = Restricted, Called = C, CalledPartyPI = Restricted, OriginalCalled = B, OrigCalledPartyPI = Restricted, LR = B, LastRedirectingPartyPI = Restricted	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = OFFERING dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwCallerIDFlags = LINECALLPARTYID_ BLOCKED dwCallerID = NP dwCallerIDName = NP dwCalledID = NP dwCalledIDName = NP dwConnectedIDFlags = LINECALLPARTYID_ BLOCKED dwConnectedID = NP dwConnectedIDName = NP dwRedirectingID = B dwRedirectingIDName = B's name dwRedirectionIDFlags = LINECALLPARTYID_ BLOCKED dwRedirectionID = NP dwRedirectionIDName = NP

Redirect Set Original Called (TxToVM)

The following table describes the message sequences for Redirece Set Original Called (TxToVM) feature where A calls B, B answers, and A and B are connected.

Table 122: Message Sequences for Redirect Set Original Called (TxToVM)

Action	CTI messages	TAPI messages	TAPI structures
Party B does lineDevSpecific for REDIRECT_SET_ ORIG_CALLED with DestDN = C's VMP and SetOrigCalled = C	Party A		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CalledChanged = True, Called = C, OriginalCalled = NULL, LR = NULL, Cause = Redirect	LINE_CALLINFO, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = CONNECTEDID, REDIRECTINGID, REDIRECTIONID	TSPI LINECALLINFO dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = NP dwRedirectionID = NP
	Party B		
	CallStateChangedEvent, CH = C2, State = Idle, reason = DIRECT, Calling = A, Called = B, OriginalCalled = B, LR = NULL	TSPI: LINE_CALLSTATE, hDevice = hCall-1, dwCallbackInstance = 0, dwParam1 = IDLE dwParam2 = 0 dwParam3 = 0	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NULL dwRedirectingID = NULL dwRedirectionID = NULL
	Party C's VMP		
	NewCallEvent, CH = C3, origin = Internal_Inbound, reason = Redirect, Calling = A, Called = C, OriginalCalled = C, LR = B	TSPI: LINE_APPNEWCALL hDevice = C dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	TSPI LINECALLINFO dwOrigin = INTERNAL dwReason = REDIRECT dwCallerID = A dwCalledID = C dwConnectedID = NULL dwRedirectingID = B dwRedirectionID = C's VMP

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

Refer and Replace Scenarios

In-Dialog Refer -Referrer in Cisco Unified Communications Manager Cluster

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer where referer is in Cisco Unified Communications Manager cluster.

Table 123: Message Sequences for In-Dialog Refer -Referrer in Cisco Unified Communications

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL		NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} LINECALLSTATE_OFFERING TAPI CallInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = “” dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referee (B)	CallState/CallInfo @Refer-to-Target (C)
C answers the call, and Refer is successful	LINECALLSTATE_IDLE with extended REFER reason	CallPartyInfoChangedEvent @ B with {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = A dwRedirectionID = C dwConnectedID = C dwReason = DIRECT dwOrigin = LINECALL ORIGIN_INTERNAL	LINECALLSTATE_CONNECTED TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = B dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

In-Dialog Refer Where ReferToTarget Redirects the Call in Offering State

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer where ReferToTarget redirects the call in Offering state.

Table 124: Message Sequences for In-Dialog Refer Where ReferToTarget Redirects the Call In

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referee (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	B gets CPIC with (calling = B, called = C, ocdpn = C, LRP = A, reason = REFER, call state = Ringback) TAPI CallInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = null dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} LINECALLSTATE_OFFERING TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = null dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL
C Redirects the call to D in offering state, and D answers	LINECALLSTATE_IDLE with extended reason = REFER (REFER considered as successful when D answers)	CallPartyInfoChangedEvent @ B with {calling = B, called = D, LRP = C, origCalled = C, reason = Redirect} Callstate = connected TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = C dwRedirectionID = D dwConnectedID = D dwReason = DIRECT dwOrigin = LINECALL ORIGIN_INTERNAL	IDLE with reason = Redirect TAPI LINECALLSTATE_IDLE D will get NewCallEvent with reason = Redirect call info same as B's call info. (calling = B, called = D, ocdpn = C, LRP = C, reason = redirect) Offering/accepted/connected

In-Dialog Refer Where Refer Fails or Refer to Target Is Busy

The following table describes the message sequences for the Refer and Replaces scenario of in-dialog refer fails or refer to target is busy.

Table 125: Message Sequences for In-Dialog Refer Where Refer Fails or Refer to Target Is Busy

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referee (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B,) and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	A-->B has a call in connected state. The call party information at A should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	A-->B has a call in connected state. The call party information at B should be {calling = A, called = B, LRP = null, origCalled = B, reason = direct} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	
(A) initiates REFER (B) to (C)	A gets LINECALLSTATE_ UNKNOWN | CLDSMT_ CALL_WAITING_STATE with extended reason = REFER TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = B dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	No change	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
C is busy / C does not answer	A gets LINECALLSTATE_CONNECTED with extended reason = REFER (REFER considered as **failed)**	If B goes to ringback when call is offered to C (C does not answer finally) it should also receive Connected Call State and CPIC event TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = Direct dwOrigin = LINECALL ORIGIN_INTERNAL	

Out-of-Dialog Refer

The following table describes the message sequences for the Refer and Replaces scenario of Out-of-Dialog Refer.

Table 126: Message Sequences for Out-of-Dialog Refer

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Referrer (A), Referee (B), and Refer-to-Target (C) exist in Cisco Unified Communications Manager cluster, and CTI is monitoring those lines	There is no preexisting call between A and B.	There is no preexisting call between A and B.	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
A initiates REFER B to (C)		B should get NewCallEvent with call info as {calling = A, called = B, LRP = null, origCalled = B, reason = REFER} TAPI CallInfo dwCallerID = A dwCalledID = B dwRedirectingID = null dwRedirectionID = null dwConnectedID = A dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_EXTERNAL	
B answers		Call state = connected (media does not flow between A and B when call goes to connected state) TAPI CallInfo (no change)	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Cisco Unified Communications Manager redirects the call to C		CallPartyInfoChangedEvent @ B with {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} TAPI callInfo dwCallerID = B dwCalledID = B dwRedirectingID = A dwRedirectionID = C dwConnectedID = C dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_EXTERNAL	NewCallEvent should be {calling = B, called = C, LRP = A, origCalled = C, reason = REFER} This info is exactly same as though caller (A) performed REDIRECT operation (except the reason is different here). TAPI callInfo dwCallerID = B dwCalledID = C dwRedirectingID = A dwRedirectionID = C dwConnectedID = B dwReason = LINECALL REASON_UNKNOWN with extended REFER dwOrigin = LINECALL ORIGIN_INTERNAL

Invite with Replace for Confirmed Dialog

The following table describes the message sequences for the Refer and Replaces scenario of invite with replace for confirmed dialog. Here, A, B, and C exist inside Cisco Unified Communications Manager. A confirmed dialog occurs between A and B. C initiates Invite to A with replace B's dialog ID.

Table 127: Message Sequences for Invite with Replace for Confirmed Dialog

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Confirmed dialog occurs between A and B	Call State = connected, Caller = A, Called = B, Connected = B, Reason = direct, gcid = GC1	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC1	

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
C Invites A by replacing B's dialog			NewCall at C gcid = GC2, reason = REPLACEs, Call state = Dialing, Caller = C, Called = null, Reason = REPLACEs
Cisco Unified Communications Manager joins A and C in a call and disconnects call leg @ B	GCID Changed to GC2, Reason = REPLACEs CPIC Caller = C, Called = A, ocdpn = A, LRP = B Reason = REPLACEs Callstate = connected TAPI callinfo caller = C, called = B, connected = C, redirecting = B, redirection = A, reason = DIRECT with extended REPLACEs, callID = GC2	Call State = IDLE, extended reason = REPLACEs	CPIC changed Caller = C, Called = A, ocdpn = A, LRP = B, Reason = REPLACEs CallState = connected TAPI callinfo Caller = C, Called = A, Connected = A, Redirecting = B, Redirection = A, reason = UNKNOWN with extended REPLACEs, callID = GC2

Refer with Replace for All in Cluster

The following table describes the message sequences for the Refer and Replaces scenario of refer with replace for all in cluster. Here, a confirmed dialog exists between A and B and A and C. A initiates Refer to C with replace B's dialog ID.

Table 128: Message Sequences for Refer with Replace for All in Cluster

Actions	CallState/CallInfo @Referrer (A)	CallState/CallInfo @Referree (B)	CallState/CallInfo @Refer-to-Target (C)
Dialog between A and B and dialog between A and C	Call State = onhold, GC1, Caller = A, Called = C, Connected = C, Reason = direct CallState = connected, GC2, Caller = A, Called = B, Connected = B, Reason = direct	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC2	Call State = connected Caller = A, Called = C, Connected = A, Reason = direct, gcid = GC1
A completes Refer to C replacing A->B's dialog (B is referred to target)	From CTI (callState = IDLE with reason = TRANSFER) TAPI call state IDLE with Reason = DIRECT with extended reason TRANSFER	GCID changed from CTI reason = TRANSFER CPIC Changed from CTI Caller = B, Called = C, Origcalled = C, LRP = A, Reason = TRANSFER TAPI callinfo Caller = B, Called = B, Connected = C, Redirecting = A, Redirection = C, Reason = DIRECT with extended reason TRANSFER. CallId = GC1	CPIC Changed from CTI with Caller = B, Called = C, Origcalled = C, LRP = A, Reason = TRANSFER TAPI callinfo caller = B, called = C, connected = B, redirecting = A, redirection = C, reason = direct with extended TRANSFER. callId = GC1

Refer with Replace for All in Cluster Replace Dialog Belongs to Another Station

The following table describes the message sequences for the Refer and Replaces scenario of refer with replace for all in cluster, where replace dialog belongs to another station. In this scenario:

A is Referrer, D is Referee, and C is Refer-to-Target.

A confirmed dialog exists between A(d1) and B & C(d2) and D.

A initiates Refer to D on (d1) with Replaces (d2).

Table 129: Message Sequences for Refer with Replace for All in Cluster, Replace Dialog Belongs to Another Station

Actions	**CallState/CallInfo @Referrer (A)**	**CallState/CallInfo @B**	**CallState/CallInfo @Refer-to-Target (C)**	**CallState/CallInfo @Referree (D)**
Dialog between A and B and dialog between C and D	Call State = onhold, Caller = A, Called = B, Connected = B, Reason = direct, gcid = GC1	Call State = connected Caller = A, Called = B, Connected = A, Reason = direct, gcid = GC1	Call State = connected Caller = C, Called = D, Connected = D, Reason = direct, gcid = GC2	Call State = connected Caller = C, Called = D, Connected = C, Reason = direct, gcid = GC2
A initiates Refer to D on (d1) with Replaces (d2)	From CTI (callState = IDLE with reason = REFER) TAPI call state IDLE with reason = DIRECT with extended reason = REFER	CPIC Changed from CTI Caller = B, Called = C, Origcalled = D, LRP = C, Reason = REPLACEs TAPI callinfo Caller = B, Called = B, Connected = D, Redirecting = C, Redirection = D, Reason = DIRECT with extended REPLACEs, CallId = GC1	From CTI (callState = IDLE with reason = REPLACEs.) TAPI call state IDLE with reason = DIRECT with extended reason = REPLACEs	GCID changed from CTI to GC1 CPIC Changed from CTI with Caller = B (referee), Called = D, Origcalled = D, LRP = C, Reason = REPLACEs TAPI callinfo caller = B, called = D, connected = B, redirecting = C, redirection = D, reason = DIRECT with extended REPLACEs, callId = GC1

Secure Conferencing

Conference with All Parties as Secure

The conference bridge includes security profile. MOH is not configured. A, B, and C get registered as Encrypted.

Action	CTI messages	TAPI messages	TAPI structures
A calls B; B answers the call	Party A		
	CallStateChangedEvent, CH = C1, GCH = G1, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = A, CH = C1 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party B		
	CallStateChangedEvent, CH = C2, GCH = G1, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
B does lineSetUpConference	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = A dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
B calls C; C answers the call	Party B		
	CallStateChangedEvent, CH = C3, GCH = G2, Calling = A, Called = B, OrigCalled = B, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = B, CH = C3 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T2 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = B dwCalledID = C dwConnectedID = C dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C4, GCH = G2, Calling = B, Called = C, OrigCalled = C, LR = NP, State = Connected, Origin = OutBound, Reason = Direct SecurityStaus = NotAuthenticated CtiCallSecurityStatusUpdate LH = C, CH = C4 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = C dwCallID = T2 dwOrigin = INTERNAL dwReason = DIRECT dwCallerID = B dwCalledID = C dwConnectedID = B dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted
B completes conf	Party B		
	CtiCallSecurityStatusUpdate LH = B, CH = C2 SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectingID = NP Devspecific Data : CallSecurityInfo = Encrypted

Hold or Resume in Secure Conference

Conference bridge includes security profile. MOH gets configured. A, B, and C represent secure phones and exist in conference with overall call security status as secure.

Action	CTI messages	TAPI messages	TAPI structures
A does lineHold	Party A		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
	Party B		
	CtiCallSecurityStatusUpdate, LH = B, CH = C2, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated
	Party C		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = NotAuthenticated	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = NotAuthenticated
A does lineResume	Party A		
	CtiCallSecurityStatusUpdate, LH = A, CH = C1, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = A dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party B		

Action	CTI messages	TAPI messages	TAPI structures
	CtiCallSecurityStatusUpdate, LH = B, CH = C2, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = B dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = B dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted
	Party C		
	CtiCallSecurityStatusUpdate, LH = C, CH = C4, SecurityStaus = Encrypted	LINE_CALLDEVSPECIFIC hDevice = C dwCallbackInstance = 0 dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA dwParam2 = SLDST_CALL_SECURITY_STATUS dwParam3 = 0	LINECALLINFO (hCall-1) hLine = dwCallID = T1 dwOrigin = CONFERENCE dwReason = UNKNOWN dwCallerID = NP dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP Devspecific Data : CallSecurityInfo = Encrypted

Secure Monitoring and Recording

Silent Monitoring

Set up:

User is in “Allow Monitoring” Group

BIB on B is set to ON

A, A1 – Customer Phones

B, B1– Agent phones

C, C1 – Supervisor phones

All Lines are Opened with Ext Version – 0x000A0000

Action	Expected result
LineInitialize. Device A,B and C is Non-Secure	
LineOpen on A,B and C	Silent Monitored Call is created in Non-Secure Mode
A calls B;B answers the Call	
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = CallAttributeInfo) will be fired to B and C
LineGetCallInfo on B	CallReason = LINECALLREASON_DIRECT CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeType = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated
LineGetCallInfo on C	CallReason = LINECALLREASON_DIRECT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info Extended Call Reason = "CtiReasonSilentMonitoring" CallAttributeType = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Action	Expected result
Varaint 1 : Monitor Customer, Agent and Supervisor Lines after Monitoring Session is Started. **Note** Start Monitoring Lines from Other Application or Close Agent and Supervisor and Reopen the same. LineGetCallInfo on B	CallReason = LINECALLREASON_UNKNOWN

Basic Silent Monitoring Scenario in Secure Mode

Action	Expected result
LineInitialize. Device A,B and C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input. LineGetCallInfo on B LineGetCallInfo on C	Silent Monitored Call is created in Secure Mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C. SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Encrypted SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Silent Monitoring Scenario on Non-Secure Call in Secure Mode

Action	Expected result
LineInitialize. Device A is not Secure Device B and C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is non Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	Monitoring Session will be started and the Media is setup in Secure Mode Events delivered will be same as use case 8.13.6.2. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C.
LineGetCallInfo on B	SRTP info is not Available security Indicator = MEDIA_NOT_ENCRYPTED CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated
LineGetCallInfo on C	SRTP info will be available security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated Same Events as above
Variant : A is Secure Call on A is Hold and Non-Secure MOH is Inserted	

Silent Monitoring Scenario on Non-Secure Call From Supervisor Which Is Secure

Action	Expected result
LineInitialize. Device A and B is not Secure Device C is Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is non Secure	Call between B and C will be Non-Secure
	No SRTP Events will be fired
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	
LineGetCallInfo on C	CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info security Indicator = MEDIA_NOT_ENCRYPTED CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Silent Monitoring Scenario on Secure Call From Supervisor Which Is Non-Secure

Action	Expected result
LineInitialize. Device A and B is Secure Device C is Not Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	• New Call will be Fired on C. • Call on C will go to Disconnected State • Request fails with new Error Code LINEERR_SECURITY_CAPABILITIES_MISMATCH. **Note** Request fails as the Supervisor Security Capabilities doesn't meet or exceed the Security status of Agent (B)

Transfer of Monitored Call From Supervisor to Other Supervisor

Action	Expected result
LineInitialize. Device A,B and C is Secure Device C1 is not Secure LineOpen on A,B,C and C1 A calls B;B answers the Call A to B call is Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	Call between B and C will be in Secure Mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C
LineGetCallInfo on C	SRTP info will be available security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted
lineDevSpecifc(CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C	LINE_REPLY (dwRequestId, 0) is returned CallSecurityStatus = Encrypted

Action	Expected result
C Transfers to C1	Transfer is successful and Monitoring Session will be Terminated. Call on C1 will be Disconnected with new Cause Code. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE. Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C.
Variant : C1 is Secure	Transfer is successful and Monitoring Session will not be disturbed. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C1 SRTP info will be available
LineGetCallInfo on B	CallAttributeInfo in devspecific part of LineCallInfo of B will contain C1's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C1's DN, Partition = C1's Partition Device Name = C1's Device Name Transaction ID = XXXX Call Security Status = Encrypted SRTP info will be available
LineGetCallInfo on C1	CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Transfer of Call From One Customer to Other

Action	Expected result
LineInitialize. Device A,B and C is Secure Device A1 is not Secure LineOpen on A,B,C and A1	Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C
A calls B;B answers the Call A to B call is Secure	Call between B and C will be in Secure Mode
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input.	Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C.
A Transfers to A1	Transfer is successful and Monitoring Session isn't disturbed.
LineGetCallInfo on B	Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = SLDST_SECURITY_STATUS_INFO) will be fired to B and C. SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C1's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Not Authenticated
LineGetCallInfo on C	SRTP info will be available Security Indicator = MEDIA_ENCRYPT_KEYS_AVAILABLE CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Park on Supervisor

Action	Expected result
LineInitialize Device A,B and C is Secure Device C1 non secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure	
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input	Call between B and C is setup with Secure mode Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C. Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C.
C parks the call	Park Operation is successful and overCallSecurity Status is degraded to Not-Authenticated
lineDevSpecifc(CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C	LINE_REPLY (dwRequestId, 0) is returned
C1 Unparks the call	UnPark operation is Successful and Monitoring session is terminated. Call on C1 is disconnected as C1doesn't have Secure Capabilities. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C.
Varaint : if LineDevSpecific for receiving Terminated Event is not set	Terminated Event is not Reported

Silent Monitoring on Conferenced Call

Action	Expected result
LineInitialize Device A and B1 is not Secure Device C and B is Secure LineOpen on A,B,B1 and C	
A, B and B1 are in Conference	Silent Monitoring Call between B and C is setup with Secure mode.
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input	Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C. Call Security Status = Not Authenticated

Conference on Monitored Call

Action	Expected result
LineInitialize. Device A, B and C is not Secure Device C1 is Secure LineOpen on A,B,C and D A calls B;B answers the Call C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input C creates conference with C1	
LineGetCallInfo on B	
LineGetCallInfo on C	
LineGetCallInfo on C1	

Action	Expected result
	Monitoring Request is successful and the Session is started Conference is created with A , C and C1 Line_CallDevSpecific (dwparam1 = DevSpecifcData, dwparam2 = OverallSecurityStatus) will be fired to C1. Call Security Status = Not Authenticated SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain CFB's info. CallAttributeTye = CallAttribute_SilentMonitorCall Call Security Status = Not Authenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated

Conference on Monitored Call

Action	Expected result
LineInitialize Device A, B and C is Secure Device C1 is not Secure LineOpen on A,B,C and C1 A calls B;B answers the Call	
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input	Monitoring Request is successful and the Session is started
lineDevSpecifc (CCiscoLineDevSpecificSetStatusMsgs) with DevSpecificStatusMsgsFlag = DEVSPECIFIC_SILENT_MONITORING_TERMINATED on C	
C creates and Completes conference with C1	Monitoring Session is ended and C and C1 will be in direct simple call. Line_CallDevSpecific will be fired for B. dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, Dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C

Supervisor Holds the Call

Action	Expected result
LineInitialize Device A, B and C is Secure Device C1 is Secure LineOpen on A,B,C and C1 C and C1 are shared lines	
A calls B; B answers the Call	
C issues LineDevSpecific (Start Monitoring) with A's permanent lineID, silent monitoring mode and NoTone as input	Monitoring session is started
C holds the call	Media will be stopped
C1 resumes the call	Media is started.Call on C will be INACTIVE (RIU Call)
Variant: C1 is not Secure and DEVSPECIFIC_SILENT_MONITORING_TERMINATED filter is enabled on C	Monitoring session is Terminated. Line_CallDevSpecific will be fired for B dwparam1 = SLDSMT_MONITORING_ENDED, dwparam2 = LINEDISCONNECTMODE_INCOMPATIBLE Call on C1 will be Disconnected with new Cause Code LINEDISCONNECTMODE_INCOMPATIBLE Line_DevSpecific (dwparam1 = SLDSMT_MONITORING_TERMINATED, dwparam2 = TransactionID – xxxx, dwparam3 = LINEDISCONNECTMODE_INCOMPATIBLE) will be fired for C

Recording

Set up

User is in Allow Recording group

A is Customer Device

B is Agent

C is Recording Device

BIB on B is set to on.

Recording Type on B is Application Invoked

C is configured as the recording device for B

Basic Recording Scenario

Action	Expected result
LineInitialize Device A,B and C is not-Secure LineOpen on A and B A calls B;B answers the Call	
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B
LineGetCallInfo on B	Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to on B CallReason = LINECALLREASON_DIRECT Devspecific part will contain the following CallAttributeTye = ‘CallAttribute_RecordedCall’ Address = C’s DN, Partition = C’s Partition Device Name = C’s Device Name Transaction ID = 0 Call Security Status = Not Authenticated
Variant 1 : Monitor the Customer and Agent Lines after the Recording Session is Started. LineGetCallInfo on B	CallReason = LINECALLREASON_UNKNOWN

Basic Recording Scenario in Secure Mode

Action	Expected result
LineInitialize Device A,B and C is Secure LineOpen on A and B A calls B;B answers the Call	
A to B call is Secure	Recording session is started in secure mode
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired on B
LineGetCallInfo on B	SRTP info will be available (for A-B Call) Devspecific part will contain the following: CallAttributeTye = CallAttribute_RecordedCall Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = 0 Call Security Status = Encrypted

Recording Scenario on Non-Secure Call in Secure Mode

Action	Expected result
LineInitialize Device A is not Secure Device B and C is Secure LineOpen on A and B A calls B;B answers the Call	
A to B call is non Secure	Recording session is started in secure mode
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B.
LineGetCallInfo on B	Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B SRTP Info is not available Devspecific part will contain the following: CallAttributeTye = CallAttribute_RecordedCall Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = 0 Call Security Status = Not Authenticated

Recording Scenario on Non-Secure Call Using Secure Recording Profile/Device

Action	Expected result
LineInitialize Device A and B is Secure Device C is Not Secure LineOpen on A,B and C A calls B;B answers the Call A to B call is Secure	
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Recording Request will Fail with existing error code LINEERR_OPERATIONFAILED **Note** Recording Failed as the Recording Device Security Capabilities doesn't meet or exceed the Security status of B

Recording Scenario When Agent Holds the Call

Action	Expected result
LineInitialize Device A and B is not Secure Device C is Secure LineOpen on A and B A calls B;B answers the Call	
A to B call is non Secure	Recording Session is started
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B Call between B and C will be Non-Secure Media between B and C is ended
LineHold on Call on B	Line_CallDevSpecific (dwparam1 = RecordingEnded) will be fired for B
B resumes the Call	Recording Session will be started Media between B and C is started Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B
Note Recording option – Automatic Call Recording Enabled	
B Resumes the Call	Recording Session will be started

Recording and Monitoring

This section describes Silent Monitoring and Recording on Agent Call in Secure Mode.

Both Silent Monitoring and Recording on Agent Call in Secure Mode

Action	Expected result

Action	Expected result
LineInitialize Device A,B,C and D are Secure D is configured as Recording Device on B LineOpen on A,B and C A calls B;B answers the Call	
A to B call is Secure	Silent Monitored Call is created in Secure Mode
C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input	C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as inputLine_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B. New call will be fired on C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C Line_CallDevSpecific(dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) will be fired for the call on C
LineGetCallInfo on B	SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Encrypted
LineGetCallInfo on C	SRTP info will be available CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Encrypted

Action	Expected result
B issues LineDevSpecific (Start Recording, BothLocalAndRemote) for A-B call	Recording session is started in secure mode Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for B. Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired on B
LineGetCallInfo on B	SRTP info will be available (SRTP info for the call Between B and A) Devspecific part will contain the following: CallAttributeTye = ‘CallAttribute_SilentMonitorCall’ Address = C’s DN, Partition = C’s Partition Device Name = C’s Device Name Transaction ID = XXXX CallAttributeTye = CallAttribute_RecordedCall Address = D’s DN, Partition = D’s Partition Device Name = D’s Device Name Transaction ID = 0 Call Security Status = Encrypted

Recording Silent Monitored Call on Supervisor

Action	Expected result
LineInitialize Device A and B is not Secure Device C and D is Secure D is the Recording Device D is configured as Recording on C LineOpen on A, B and C A calls B;B answers the Call A to B call is non Secure C issues LineDevSpecific (Start Monitoring) with B's permanent lineID, silent monitoring mode and NoTone as input LineGetCallInfo on B LineGetCallInfo on C C issues LineDevSpecific (Start Recording, BothLocalAndRemote) for B-C call	

Action	Expected result
	Line_CallDevSpecific (dwparam1 = MonitoringStarted) will be fired for B New call will be fired on C (Silent Monitoring call) Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to B and C SRTP info will not be available CallAttributeInfo in devspecific part of LineCallInfo of B will contain C's info. CallAttributeTye = 'CallAttribute_SilentMonitorCall' Address = C's DN, Partition = C's Partition Device Name = C's Device Name Transaction ID = XXXX Call Security Status = Unauthenticated SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeInfo in devspecific part of LineCallInfo of C will contain B's info CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallSecurityStatus = Not Authenticated Recording Session is started Line_CallDevSpecific (dwparam1 = RecordingStarted) will be fired for C Line_CallDevSpecific(dwparam1 = DevSpecifcData, dwparam2 = CallAttributeInfo) will be fired to C

Action	Expected result
LineGetCallInfo on C	SRTP info will not be available Security Indicator = MEDIA_NOT_ENCRYPT CallAttributeTye = CallAttribute_SilentMonitorCall_Target Address = B's DN, Partition = B's Partition Device Name = B's Device Name Transaction ID = XXXX CallAttributeTye = 'CallAttribute_RecordedCall' Address = D's DN, Partition = D's Partition Device Name = D's Device Name Transaction ID = 0 Call Security Status = Not Authenticated

Shared Lines-Initiating a New Call Manually

The following table describes the message sequences for Shared Lines-Initiating a new call manually where Party A and Party A' represent shared line appearances. Also, Party A and Party A' are idle.

Action	CTI messages	TAPI messages	TAPI structures
1. Party A goes off-hook	NewCallEvent, CH = C1, GCH = G1, Calling = A, Called = NP, OrigCalled = NP, LR = NP, State = Dialtone, Origin = OutBound, Reason = Direct, RIU = false	LINE_APPNEWCALL hDevice = A dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-1 dwParam3 = OWNER	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALTONE dwParam2 = UNAVAIL dwParam3 = 0	No change
1. Party A goes off-hook	Party A’		
	NewCallEvent, CH = C1, GCH = G1, Calling = A’, Called = NP, OrigCalled = NP, LR = NP, S tate = Dialtone, Origin = OutBound, Reason = Direct, RIU = true	LINE_APPNEWCALL hDevice = A’ dwCallbackInstance = 0 dwParam1 = 0 dwParam2 = hCall-2 dwParam3 = OWNER	LINECALLINFO (hCall-2) hLine = A’ dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A’ dwCalledID = NP dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Dialtone, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0	No change

Action	CTI messages	TAPI messages	TAPI structures
2. Party A dials Party B	Party A		
	CallStateChangedEvent, CH = C1, State = Dialing, Cause = CauseNoError, Reason = Direct, Calling = A, Called = NP, OrigCalled = NP, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = DIALING dwParam2 = 0 dwParam3 = 0	No change
	Party A’		
	None	None	None
3. Party B accepts call	Party A		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A, CalledChanged = true, Called = B, Reason = Direct, RIU = false	Ignored	No change

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = PROCEEDING dwParam2 = 0 dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CALLERID, CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = RINGBACK dwParam2 = 0 dwParam3 = 0	No change
3. Party B accepts call (continued)	Party A’		
	CallPartyInfoChangedEvent, CH = C1, CallingChanged = False, Calling = A’, CalledChanged = true, Called = B, Reason = Direct, RIU = true	Ignored	No change

Action	CTI messages	TAPI messages	TAPI structures
	CallStateChangedEvent, CH = C1, State = Proceeding, Cause = CauseNoError, Reason = Direct, Calling = A', Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CALLERID, CALLEDID dwParam2 = 0 dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A' dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A' dwCalledID = B dwConnectedID = NP dwRedirectionID = NP dwRedirectionID = NP
	CallStateChangedEvent, CH = C1, State = Ringback, Cause = CauseNoError, Reason = Direct, Calling = A', Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0	No change
4. Party B answers call	Party A		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A, Called = B, OrigCalled = B, LR = NP, RIU = false	LINE_CALLSTATE hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = ACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-1 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0, dwParam3 = 0	LINECALLINFO (hCall-1) hLine = A dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

Action	CTI messages	TAPI messages	TAPI structures
	Party A’		
	CallStateChangedEvent, CH = C1, State = Connected, Cause = CauseNoError, Reason = Direct, Calling = A’, Called = B, OrigCalled = B, LR = NP, RIU = true	LINE_CALLSTATE hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTED dwParam2 = INACTIVE dwParam3 = 0 LINE_CALLINFO hDevice = hCall-2 dwCallbackInstance = 0 dwParam1 = CONNECTEDID dwParam2 = 0, dwParam3 = 0	LINECALLINFO (hCall-2) hLine = A’ dwCallID = T1 dwOrigin = OUTBOUND dwReason = DIRECT dwCallerID = A’ dwCalledID = B dwConnectedID = B dwRedirectionID = NP dwRedirectionID = NP

SRTP

Media Terminate by Application (Open Secure CTI Port or RP)

- Negotiate version
- Sends LineOpen with extension version as 0x8007000
- Send CciscoLineDevSpecificUserSetSRTPAlgorithmID
- Send CCiscoLineDevSpecificUserControlRTPStream
- Now, the CTI port or RP gets registered as secure port
- Make call from secure IP phone to the CTI port or RP port
- Answer the call from application
- SRTP indication gets reported as LineDevSpecific event
- SRTP key information get stored in LINECALLINFO::devSpecifc for retrieval

Media Terminate by TSP Wave Driver (Open Secure CTI Port)

- Negotiate version
- Sends LineOpen with extension version as 0x4007000
- Send CciscoLineDevSpecificUserSetSRTPAlgorithmID
- Send CciscoLineDevSpecificSendLineOpen

- Now, the CTI port gets registered as secure port
- Make call from secure IP phone to the CTI port
- Answer the call from application
- SRTP indication gets reported as LineDevSpecific event
- SRTP key information get stored in LINECALLINFO::devSpecifc for retrieval

Support for Cisco IP Phone 6900 Series

Use cases related to Cisco Unified IP Phone 6900 Series support feature are mentioned below:

Monitoring Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 Phone with Roll Over Mode User is added to New User Group. Application does Line Initialize
Expected Results	Lines on the Cisco Unified IP Phone 7931 will be enumerated. Application would be able to Open Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 and it would be able to control and perform call operations on phone.

Monitoring Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode Step 1: Application does Line Initialize Step 2: User is added to New User Group.
Expected Results	Step 1: Lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 will not be enumerated Application will not be notified about the device A and it will not be able to monitor. Step 2: Application will be receiving PHONE_CREATE and LINE_CREATE events for the Device and lines on that Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode. Now Applications would be able to Monitor and control Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931.

Transfer Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to new user Group.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A. Variants: Application Opens only Line A on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931
Expected Results	Call on A will go to OnHold State. New call will be created on Line B. Application then has to complete Transfer using DTAL feature. Variants: Applications would not be able to Complete Transfer from Application as the Line B is not monitored.

Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Conference scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group.

Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D are two SCCP phones Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize C calls A,A answers SetupConference on A.
Expected Results	Call on A will go to OnHold State. New call will be created on Line B. Application then has to complete Conference using Join Across Lines feature.

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/OnHoldPendingConference. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with Conference LineCompleteTransfer with Mode as Conference to complete Conference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -Roll Over to any Line Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/OnHoldPendingConference. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with Conference LineCompleteTransfer with Mode as Conference to complete Conference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.

Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	SetupTransfer Request will fail with error "LINEERR_CALLUNAVAIL".
Variants	Test the same Scenario with SetupConference

Transfer/Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Transfer/Conference Scenario on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN Max Number of Calls: 2 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers SetupTransfer on A.
Expected Results	Call on A will go to OnHoldPendingTransfer/Conference State. New Consult call will be created on Line A. Application then has to complete Transfer using CompleteTransfer or DTAL feature.
Variants	Test the same Scenario with SetupConference

LineMakeCall Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing LineMakeCall Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN" or "Roll Over to Any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers LineMakeCall on A.
Expected Results	LineMakeCall Operation will fail with error "LINEERR_CALLUNAVAIL". Roll Over Doesn't Happen to second line as the roll over is only for Outbound Calls.

LineUnPark Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing LineUnPark Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Lines A and B are configured with Different DN Outbound Roll Over Mode -Roll Over within same DN" or "Roll Over to Any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers LineUnPark on A.(tires to retrieve the available Parked Call from Park DN)

Expected Results	LineUnPark Operation will fail with error "LINEERR_CALLUNAVAIL". Roll Over Doesn't Happen to second line as the roll over is only for Outbound Calls.

EM Login/Logout Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing EM Log In/Out Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. EM Profile is logged onto the Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Test the Use Case from UseCase#1 to UseCase#10
Expected Results	Same as the Use Case tested.

Manual Transfer Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Existing Call Events on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 when User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -Roll Over to any Line Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press Transfer Button on Cisco Unified IP Phone 6900 Series and Dial D. Step 4: Answer the Call on D Step 5: Complete Transfer from Phone A Variant: Monitor Phones after Transfer is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Call will be on ONHOLDPENDINGTRANSFER state. Step 5: Both the calls on A and B will go to IDLE state. C and D will be in Simple Call. Variant: Same as this Use Case

Manual Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press conference Button on Cisco Unified IP Phone 6900 Series and Dial D. Step 4: Answer the Call on D Step 5: Complete Conference from Phone Variant: Monitor Phones after Conference is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Conference Model is created as today on Non-Cisco Unified IP Phone 6900 Series. Step 5: A ,C and D will be in conference Conference model will be created on Line A. Variant: Same as this Use Case.

Manual Conference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 1: From Phone C call A Step 2: Answer the Call on A Step 3: Press conference Button on Cisco Unified IP Phone 6900 Series Phone and Dial D. Step 4: Answer the Call on D Step 5: Complete Conference from Phone Variant: Monitor Phones after Conference is completed from Phone.

Expected Results	Step 4: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. **Note** When consult call is created on the same Line; Conference Model is created as today on Non-Cisco Unified IP Phone 6900 Series Phone. Step 5: A ,C and D will be in conference Conference model will be created on Line A. Variant: Same as this Use Case.

SetupConference Operation on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 behavior When User is added to New User Group and different Roll Over Mode.
Test Setup	User is added to New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Roll Over to any Line" Max Number of Calls: 1 Busy Trigger : 1 Application does Line Initialize; Application opens all the lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. C calls A,A answers Step 1: SetupTransfer on A. Step 2: Complete Conference From Phone.
Expected Results	Step 1: Call on Line A will be in OnHold State. Call on Line B will be in Connected State. Step 5: A ,C and D will be in conference Conference model will be created on Line A.

BWC on Cisco Unified IP Phone 7931 in Non Roll Over Mode When User Is Removed From New User Group

Description	Testing Cisco Unified IP Phone 7931 Phone behavior in Non Roll Over Mode When User is removed from New User Group.

Test Setup	User is Removed from New User Group. A,B are two lines on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Non-Roll Over Mode C, D is two SCCP phones. Outbound Roll Over Mode -"Non Roll Over Mode" Max Number of Calls: 1 Busy Trigger: 1 Application does Line Initialize
Expected Results	Lines on the Cisco Unified IP Phone 7931 will be enumerated. Application would be able to Open Cisco Unified IP Phone 7931 with Non-Roll Over Mode and it would be able to control and perform call operations on Phone.

Acquire Device on Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode When User Is Added to New User Group

Description	Testing Behavior of Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 on Super Provider when User is added to new user Group.
Test Setup	A -Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 with Roll Over Mode User is Added to New User Group. Step 1: Application does Line Initialize Step 2: LineDevSpecific to Acquire Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931. Step 3: User is removed from New User Group.
Expected Results	Step 2: Application will be receiving PHONE_CREATE and LINE_CREATE events for the Device and lines on that Cisco Unified IP Phone 6900 Series/Cisco Unified IP Phone 7931 in Roll Over Mode. Step 3: Application will be receiving LINE_REMOVE and PHONE_REMOVE for the Cisco Unified IP Phone 7931 and Application will no longer be able to monitor or control that device.

Support for Cisco Unified IP Phone 6900 and 9900 Series Use Cases

The use cases related to Support for Cisco Unified IP Phone 6900 and 9900 Series are provided below:

Check Max Calls Information

Action	Events, Requests, and Responses
Application calls LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks Max Calls field.	MaxCalls = 4 in LineDevCaps:DevSpecific

Check Busy Trigger Information

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks busy trigger field.	BusyTrigger = 2 in LineDevCaps:DevSpecific

Check Line Instance

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks line instance field.	LineInstanceNumber = 1 in LineDevCaps:DevSpecific

Check Line Label

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks line label field.	LineLable = label_2000 in LineDevCaps:DevSpecific

Check Voice Mail Pilot

.

Action	Events, Requests, and Responses
Application does LineInitialize	LineInitialize successful
Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	VoiceMailPilot = 5000 in LineDevCaps:DevSpecific

Check Registered IP Address of the Device or Line

.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks IP address field.	LineInitialize successful RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check IP address field.	PhoneInitialize successful RegisteredIPv4Address & RegisteredIPv6Address available in PhoneDevCaps:DevSpecific

Check Consult Rollover Information of the Line

ConsultRollOver is true for the device

.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks consult roll over field.	LineInitialize successful ConsultRollOver flag is true in LineDevCaps:DevSpecific
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful ConsultRollOver flag is true in PhoneDevCaps:DevSpecific.
Variance: Phone does not support rollover Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful ConsultRollOver flag is false in PhoneDevCaps:DevSpecific.

Check JAL or DTAL Information of the Line

JAL or DTAL is true for the device.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks JAT/DTAL field.	LineInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is true in LineDevCaps:DevSpecific.
Variance: Perform PhoneInitialize and check PhoneGetDevCpas to check consult roll over field.	PhoneInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is true in PhoneDevCaps:DevSpecific.
Variance: Phone does not support jal/dtal Perform PhoneInitialize and check PhoneGetDevCpas to check JAT/DTAL field.	PhoneInitialize successful JoinAcrossLine and DirectTransferAcrossLine flag is false in PhoneDevCaps:DevSpecific.

Handle Voice Mail Pilot Change

Voice Mail Pilot number is changed to 6000.

Action	Events, Requests, and Responses
Application does LineInitialize Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	LineInitialize successful VoiceMailPilot = 5000 in LineDevCaps:DevSpecific
Voice Mail Pilot number is changed to 6000.	LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating Voice Mail Pilot is changed.
Application calls LineGetDevCaps, and checks Voice Mail Pilot field.	VoiceMailPilot = 6000 in LineDevCaps:DevSpecific
Variance: also applies to Line Label	

Check IP Address When Device Is Unregistered or Registered

It is assumed that phone uses static IP address and is already registered.

Action	Events, Requests, and Responses
Application calls LineInitialize Application calls LineGetDevCaps, and checks IP address field.	Initializesuccessful RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific, and RegisteredIPAddressMode is IPAddress_IPv4_IPv6.
Reset device	Phone or line goes out of service. LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address information is changed.
Application calls LineGetDevCaps, and checks IP address field.	The same RegisteredIPv4Address & RegisteredIPv6Address available in LineDevCaps:DevSpecific, but RegisteredIPAddressMode is IPAddress_Unknown.
Device re-registered with CUCM.	Phone or line back in service. LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address information is changed.
Application calls LineGetDevCaps, and checks IP address field.	The same RegisteredIPv4Address and RegisteredIPv6Address available in LineDevCaps:DevSpecific, but RegisteredIPAddressMode is set to IPAddress_IPv4_IPv6.
Variance: Phone uses DHCP and new IP address is obtained for registering.	LineDevSpecific (SLDSMT_LINE_PROPERTY_CHANGED) indicating registered IP address is changed New IPAddress will be in devSpecific when application queries LineGetDevCap. .

Swap or Cancel

Use cases related to Swap or Cancel feature are mentioned below:

Connected Transfer

Device A, B, C where A is a Cisco Unified IP Phone (future version)..

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press transfer	For A: Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-3 DIALTONE
A picks "Active Calls"	Call-3 goes IDLE

Action	Expected events
A picks call (A‡C) and presses transfer to complete transfer	For A: Both calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Connected Transfer on Phones with Shared Lines

Device A, B, C, A' where A and A' are sharedline.

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A For A': Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-2 LINE_CALLSTATE param1 = x100, CONNECTED_INACTIVE Caller = A, Called = B Connected B

Action	Expected events
User performs connected transfer on Cisco Unified IP phone (future version)	For A and A': All calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Connected Transfer: Initiate From Phone, Complete From CTI

Device A, B, C .

Action	Expected events
A ‡ C is on hold A ‡ B is connected,	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
Application sends either CompleteTransfer or DirectTransfer on A	For A and A': All calls go IDLE For B1: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Consult Transfer: Resume Primary Call (Implicit Cancel)

Action	Expected events
A ‡ B A setup consult transfers to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press resume to resume A‡ B call	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Consult Transfer: Swap Calls

Action	Expected events
A ‡B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C

Action	Expected events
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A press "Transfer" to complete transfer	For A: Calls go IDLE For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = C For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = B

Consult Transfer on Phone: Swap Calls; CTI Sends SetupTransfer on Connected Call

Action	Expected events
A ‡ B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
Application calls LineSetupTransfer on A's connected call (A‡B) to initiate transfer	Request succeeds as phone cancels existing feature plan and allow CTI request to go through.

Consult Transfer: Swap and Cancel

Action	Expected events
A ‡ B A setup consult transfer to C And C answer	For A: Call-1 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGTRANSFER Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A press Swap	For A: The scenario will look exactly the same when resume primary call. Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses Cancel	No TSP event since it is handled during swap operation

RoundTable Connected Conference

Action	Expected events
A ‡ B A puts call on hold A creates new call to C, C answer	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Conference"	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGCONFENRENCE Caller = A, Called = C Connected C Call-3 DIALTONE

Action	Expected events
A picks active call (A‡ C) on phone UI, and presses "Conference" to complete the conference	For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = C, connected = C Call-3 IDLE For B: For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = C, connected = C For C: For A: CONNECTED CONFERENCED Caller = A, called = C, connected = C CONFERENCED Caller = C, called = B, connected = B

RoundTable Connected Conference: Cancel

Action	Expected events
A ‡ B A puts call on hold A creates new call to C, C answers	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Conference"	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, CONFERENCED Caller = A, Called = C Connected C Call-3 LINE_CALLSTATE param1 = x100, ONHOLDPENDINGCONFENRENCE Caller = A, Called = C Connected C Call-4 DIALTONE

Action	Expected events
A picks "Active Calls"	For A: Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C Call-3 / Call-4 IDLE
A presses Cancel softkey	For A: Call-1 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Set Up Consult Conference From RT, Then Swap and Complete Conference From RT

Action	Expected events
A ‡ B A sets up conference to C, C answer	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Swap"	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x100, HOLD Caller = A, Called = C Connected C

Action	Expected events
A presses "Conference" to complete conference	For A: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = A, called = C, connected = C For B: CONNECTED CONFERENCED Caller = A, called = B, connected = B CONFERENCED Caller = B, called = C, connected = C For C: For A: CONNECTED CONFERENCED Caller = A, called = C, connected = C CONFERENCED Caller = C, called = B, connected = B

Set Up Consult Conference From RT, Then Swap and Cancel From Phone with Shared Line Scenario

A and A’ are shared lines..

Action	Expected events
A ‡ B A sets up conference to C, C answers	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C For A' CONNECTED INACTIVE Caller = A, celled = B, connected = B CONNECTED INACTIVE Caller = A, celled = C, connected = C For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A
A presses "Swap"	For A: The scenario looks the same when primary call resumes Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C

Action	Expected events
A presses "Cancel"	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected = C
	For A' Call-1 LINE_CALLSTATE CONNECTED INACTIVE Caller = A, Called = B Connected = B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Set Up Consult Conference From RT: Resume Primary Call (Implicit Cancel)

Action	Expected events
A ‡ B A sets up conference to C, C answer	For A: ONHOLDPENDINGCONF CONFERENCED Caller = A, called = B, connected = B CONNECTED Caller = A, called = C, connected = C
	For A' CONNECTED INACTIVE Caller = A, celled = B, connected = B CONNECTED INACTIVE Caller = A, celled = C, connected = C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

Action	Expected events
A resumes A‡B call	For A: Call-1 LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B Connected B Call-2 LINE_CALLSTATE param1 = x400, HOLD Caller = A, Called = C Connected C
	For B: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = B, Connected = A
	For C: LINE_CALLSTATE param1 = x100, CONNECTED Caller = A, Called = C, Connected = A

User Is Removed From Standard Supports Connected Xfer/Conf Group

Action	Expected events
User is in Standard Supports Connected Xfer/Conf group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP
Remove user from "Standard Supports Connected Xfer/Conf" user group	APP receives PHONE_REMOVE / LINE_REMOVE

User Is Removed From Standard Supports Connected Xfer/Conf Group

Action	Expected events
User is in Standard Supports Connected Xfer/Conf group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP

Action	Expected events
Remove user from Standard Supports Connected Xfer/Conf user group	APP receives PHONE_REMOVE / LINE_REMOVE

User Is Removed From Standard Supports Connected Xfer/Conf Group While Line Is Open

Action	Expected events
user is in "Standard Supports Connected Xfer/Conf" group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is enumerated to APP
App sends LineOpen to open line on Cisco Unified IP phone (future version) phone	Successful
Remove user from Standard Supports Connected Xfer/Conf group	TSP sends LINE_CLOSE APP receives LINE_REMOVE

User Is Added to Standard Supports Connected Xfer/Conf Group

Action	Expected events
user is not in "Standard Supports Connected Xfer/Conf" group RT phone A is in user's control list Application does LineInitialize	RT PHONE/LINE is not enumerated to APP
Add user to Standard Supports Connected Xfer/Conf group	APP receives PHONE_CREATE / LINE_CREATE

Unrestricted Unified CM

Table 130: Application Tries Secure Connection to Unrestricted Unified CM During Upgrade

Action	Events, requests and responses
CUCM – Restricted UCM TSP is configured to connect Secure Application calls LineInitialize	 LineInitialize successful All lines associated are enumerated.
*** Upgrade CUCM to Unrestricted Unified CM CCM/CTI services restarted	OutOfService Events for all the Devices/Lines. ***TSP will internally try to Connect CTI in Secure mode. As CTI is upgraded to Non-secure, the Connection Fails and applications are not notified. Application has to disable “Secure Connection to CTI Manager” on the Security tab in TSP UI to setup connection to CTI/CUCM.

Table 131: Application Tries Secure Connection to Unrestricted Unified CM After Upgrade

Action	Events, requests and responses
CUCM – Restricted UCM TSP is configured to connect Secure Application calls LineInitialize	 LineInitialize successful All lines associated are enumerated.
Application calls LineShutdown *** Upgrade CUCM to Unrestricted UCM Application calls LineInitialize	LineShutDown successful LineInitialize successful. No lines are enumerated to application.

Table 132: Registering Secure CTI Port with Unrestricted Unified CM CTI Manager

Action	Events, requests and responses
CUCM – Unrestricted UCM Setup Non-Secure Connection Application calls LineInitialize Register CTI Port in Secure Mode • LineOpen – with Ext – 80070000 • LineDevspecific – CciscoLineDevSpecificUserSetSRTPAlgorithmID	LineInitialize successful All lines associated to end users are enumerated. LineReply – with error -LINEERR_OPERATIONUNAVAIL

Table 133: Registering Secure CTI Port with Unrestricted Unified CM CTI Manager

Action	Events, requests and responses
Setup: • Node 1 – UnRestricted UCM • Node 2 – Restricted UCM – Secure CTI Port – Device Pool – with Node 1 as High Priority CM. TSP is configured to connect to CTI Manager of Node 2. Set up Secure Connection Application calls LineInitialize Register CTI Port in Secure Mode • LineOpen – with Ext – 80070000 • LineDevspecific – CciscoLineDevSpecificUserSetSRTPAlgorithmID • LineDevSpecific -CCiscoLineDevSpecificUserControlRTPStream	LineInitialize successful All Lines Associated are Enumerated. LineReply – success LINE_CLOSE for the CTI Port

LineHold Enhancement

Prerequisites

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

- Device A (IP Phone, Line A1 (dn: 1000))
- Device B (IP Phone, Line B1 (dn: 2000))
- The content id corresponding to VoH stream is contentID1
- User1 has in its control list: Devices A and B. All devices and lines are observed
- Provider is opened (lineInitializeEx successfully executed)
- All relevant lines are opened with Extension version 0x000D0000 and in service

Table 134: Basic Case - Hold with ContentID to Be Played

Action	TAPI Messages	TAPI Structures
Create Call: LineMakeCall() on Line-A w ith DestAddress="DN of B" and B answers the Call	At A: LINE_CALLSTATE dwParam1 = 0x00000100 (CONNECTED) At B: LINE_CALLSTATE dwParam1 = 0x00000100 (CONNECTED)	CallInfo on A: CallerID: 1000 CalledID: 2000 ConnectedID: 2000
Application issues CCiscoLineDevSpecificHoldEx with ContentID = contentID1 on hCall1(call on A1) *** Call will be placed on Hold and VoH stream selected is played to B.	At A: LINE_CALLSTATE dwParam1 = 0x00000400 (LINECALLSTATE_ONHOLD)	

Whisper Coaching

Setup

Customer Phone – IP Phone A

Agent Phone – IP Phone B

Supervisor Phone – IP Phone C

Application monitoring all lines on all devices

New extension is negotiated when application opens lines

Application Initiates a Whisper Coaching Session

Service Parameter Setting: Observed Target = false, Observed Connected Parties = true

Table 135: Application Initiates a Whisper Coaching Session

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A
C issues CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_LocalOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly **Note** Media events are not received at B. At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Application Updates the Monitoring Mode

Service Parameter Setting: Observed Target = true, Observed Connected Parties = false

Table 136: Application Updates the Monitoring Mode (Silent to WhisperCoaching) and Then Updates the Monitoring Mode (WhisperCoaching to Silent)

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A
C issues CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_RECEPTION)

Action	Events, Requests, and Responses
C issues CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_BothLocalAndRemote	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Whisper_Coaching, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Whisper_Coaching, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
C issues CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Silent tone = PlayToneDirection_NoLocalOrRemote	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Agent Holds the Customer Call with Whisper Coaching Then Agent S Shared Line Resumes the Call

Additional Setup: Agent shared line IP Phone B

Table 137: Agent Holds the Customer Call with Whisper Coaching, Then Agent's Shared Line Resumes the Call

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly B holds the call	At B: ONHOLD devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At B': ONHOLD At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_STOP_RECEPTION)
B resumes the call	At B: CONNECTED At B': CONNECTED, INACTIVE At C: LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Action	Events, Requests, and Responses
B holds the call B resumes the call	At B: CONNECTED, INACTIVE LineDevSpecific(SLDSMT_MONITORING_ENDED) At B': CONNECTED At C: IDLE

Agent Transfers a Whisper Coaching Call Monitoring Call Goes Idle at the Supervisor

Additional Setup: IP Phone D

Table 138: Agent Transfers a Whisper Coaching Call, Monitoring Call Goes Idle at the Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly B setup transfer to D and D answers	At B: ONHOLDPENDTRANSFER devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = B, Called = D, Connected = D At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
B complete transfer to D	At B: IDLE IDLE At C: IDLE

Application Updates the Monitoring Mode (WhisperCoaching to Silent)

Additional Setup: IP Phone D

Table 139: Application Updates the Monitoring Mode (WhisperCoaching to Silent) After the Agent Conferences the Whisper Coaching Call

Action	Events, Requests, and Responses
A initiates Call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly B setup conference to D and D answers B complete conference to D	At B: CONFERENCE Calling = A, Called = B, Connected = B CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONFERENCE Calling = B, Called = D, Connected = D At C: CONNECTED Calling = C, Called = B/B's Name Connected = “”/Whisper, Redirection = “”/Whisper, Redirecting = “”/Whisper, devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
C issues a CciscoLineDevSpecificMonitoringUpdateMode with: mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	At B: LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONFERENCE Calling = A, Called = B, Connected = B CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONFERENCE Calling = B, Called = D, Connected = D At C: LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_MONITORING_MODE_UPDATED, MonitorMode_Silent, PlayToneDirection_RemoteOnly) CONNECTED devSpecific type = CallAttribute_SilentMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
B issues a lineRemoveFromConference to drop D.	At B: CONNECTED devSpecific type = CallAttribute_SilentMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly IDLE IDLE At C: No change in callInfo and no additional events

Supervisor Holds/Resumes the Whisper Coaching Monitoring Session

Additional Setup: IP Phone D

Table 140: Supervisor Holds/Resumes the Whisper Coaching Monitoring Session

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C holds the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: ONHOLD Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_STOP_TRANSMISION) LineDevSpecific(SLDSMT_STOP_RECEPTION)
C resumes the call	At C: CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Supervisor Transfers the Whisper Coaching Session to Another Supervisor

Additional Setup: Supervisor IP Phone D

Table 141: Supervisor Transfers the Whisper Coaching Session to Another Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C setup transfers the call to D, D answers	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: ONHOLDPENDTRANSFER Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONNECTED Calling = C, Called = D, Connected = C

Action	Events, Requests, and Responses
C complete transfers the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = D, partition = D's Partition, deviceName = D's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: IDLE IDLE At D: CONNECTED Calling = C, Called = D Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Supervisor Conferences the Whisper Coaching Session to Another Supervisor

Additional Setup: Supervisor IP Phone D

Table 142: Supervisor Conferences the Whisper Coaching Session to Another Supervisor

Action	Events, Requests, and Responses
A initiates call to B and B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly C setup conferences the call to D and D answers	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONFERENCE ONHOLDPENDCONF Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONNECTED Calling = C, Called = D, Connected = C

Action	Events, Requests, and Responses
C complete conferences the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = CFB, partition = CFB Partition, deviceName = CFB device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: CONFERENCE Calling = C, Called = B/B's Name, Connected = CFB CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CONNECTED Calling = C, Called = D, Connected = D At D: CONFERENCE Calling = C, Called = D, Connected = D CONNECTED CONNECTED Calling = D, Called = CFB, Connected = CFB

Action	Events, Requests, and Responses
C drops the call	At B: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = D, partition = D's Partition, deviceName = D's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly At C: IDLE IDLE IDLE At D: CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = B, partition = B's Partition, deviceName = B's device transactionID = xxxx, tone = PlayToneDirection_RemoteOnly
D issues a CciscoLineDevSpecificMonitoringUpdateMode with: permLineId = B permLineId mode = MonitorMode_Silent tone = PlayToneDirection_RemoteOnly	

Application Initiates a Whisper Coaching Session Second Application on a Different Client Opens All Lines

Additional Setup: Supervisor IP Phone D

Table 143: Application Initiates a Whisper Coaching Session, Second Application on a Different Client Opens All Lines

Action	Events, Requests, and Responses
A initiates Call to B, B answers C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_RemoteOnly	At B (Application 1): CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C (Application 1): CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Action	Events, Requests, and Responses
Second application opens all lines	At B (Application 2): CONNECTED devSpecific CallAttributeBitMask = TSPCallAttribute_WhisperMonitorCall type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly At C (Application 2): CONNECTED CallAttributeBitMask = TSPCallAttribute_WhisperMonitorCall_Target Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly

Secure R & M with Whisper Coaching Supports

- Overall security status of the monitoring call either silent or whisper must be same. See Secure monitoring use cases.
- Overall security status of the monitoring call must not change if monitor mode is updated either from silent to whisper or vice versa.

Application Initiates a Secure Whisper Coaching Session

Additional Setup: All devices are secure

Table 144: Application Initiates a Secure Whisper Coaching Session

Action	Events, Requests, and Responses
A initiates call to B and B answers	At A: CONNECTED Calling = A, Called = B, Connected = B At B: CONNECTED Calling = A, Called = B, Connected = A

Action	Events, Requests, and Responses
C issues a CciscoLineDevSpecificStartCallMonitoring with: permLineId = B permLineId mode = MonitorMode_Whisper_Coaching tone = PlayToneDirection_LocalOnly	At B: LineDevSpecific(SLDST_START_CALL_MONITORING) CONNECTED devSpecific type = CallAttribute_WhisperMonitorCall dn = C, partition = C's Partition, deviceName = C's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CallSecurityStatus = OverallCallSecurityStatus_Encrypted **Note** Media events are not received at B and SRTP keys are not available. At C: LineDevSpecific (dwparam1 = DevSpecificData, dwparam2 = SLDST_SRTP_INFO, dwParam3 = MEDIA_ENCRYPT_KEYS_AVAILABLE) SRTP keys are available CONNECTED Calling = C, Called = B/B's Name Connected = ""/Whisper, Redirection = ""/Whisper, Redirecting = ""/Whisper, devSpecific type = CallAttribute_WhisperMonitorCall_Target dn = B, partition = B's Partition, deviceName = B's device transactionId = xxxx, tone = PlayToneDirection_RemoteOnly CallSecurityStatus = OverallCallSecurityStatus_Encrypted LineDevSpecific(SLDSMT_START_TRANSMISION) LineDevSpecific(SLDSMT_START_RECEPTION)

Application Updates the Monitoring Mode on an Agent Call That Is on Hold

The application updates the monitoring mode on an agent call that is on hold as follows:

1. A initiates Call to B and B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:

- permLineId = B permLineId
- mode = MonitorMode_Whisper_Coaching
- tone = PlayToneDirection_RemoteOnly

3. B puts the call on hold
4. C issues CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
5. LINE_REPLY returns LINEERR_INVALCALLSTATE

Application Initiates Whisper Coaching Where the Agent Is a SIP Device with Older Firmware Version That Does Not Support Media Mixing

The application initiates Whisper Coaching where the agent is a SIP device with older firmware version that does not support media mixing as follows:

1. A initiates Call to B and B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
3. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

Application Updates the Monitoring Mode Where the Agent Is a SIP Device with Older Firmware Version That Does Not Support Media Mixing

The application updates the monitoring mode where the agent is a SIP device with older firmware version that does not support media mixing as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
3. C issues a CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly

4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

Application Updates the Monitoring Mode on a Monitoring Call at the Supervisor That Is in a Conference

The application updates the monitoring mode on a monitoring call at the supervisor that is in a conference as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Silent
   - tone = PlayToneDirection_RemoteOnly
3. C setups or completes the call to D and D answers.
4. C issues a CciscoLineDevSpecificMonitoringUpdateMode with:
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
5. LINE_REPLY returns LINEERR_OPERATIONUNAVAIL

Application Initiates Whisper Coaching on an Agent That Is Already Playing an Agent Greeting

The application initiates Whisper Coaching on a agent that already is playing an agent greeting as follows:

1. A initiates Call to Band B answers
2. B issues a CCiscoLineDevSpecificStartSendMediaToBIBRequest with:
   - DN = IVR DN
   - timeout = 30
3. C issues a CciscoLineDevSpecificStartCallMonitoring with:
   - permLineId = B permLineId
   - mode = MonitorMode_Whisper_Coaching
   - tone = PlayToneDirection_RemoteOnly
4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL

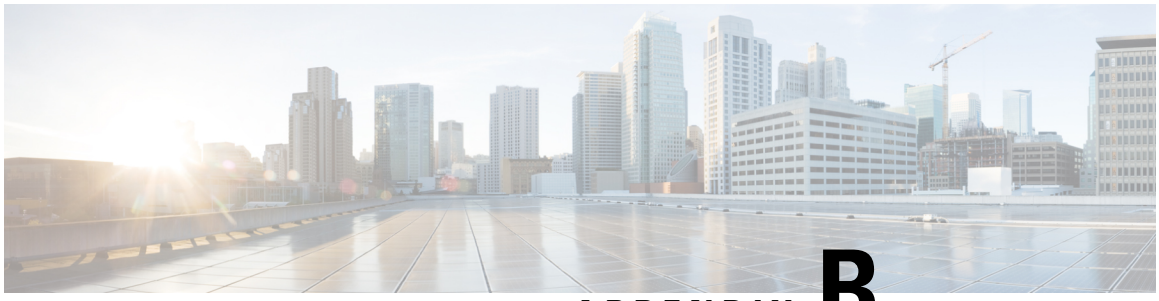
Application Initiates Agent Greeting on a Call That Already Has a Whisper Coaching Session

The application initiates Agent Greeting on a call that already has a Whisper Coaching session as follows:

1. A initiates Call to Band B answers
2. C issues a CciscoLineDevSpecificStartCallMonitoring with:

- permLineId = B permLineId
- mode = MonitorMode_Whisper_Coaching
- tone = PlayToneDirection_RemoteOnly

3. B issues a CCiscoLineDevSpecificStartSendMediaToBIBRequest with:
   - DN = IVR DN
   - timeout = 30
4. LINE_REPLY returns LINEERR_RESOURCEUNAVAIL



APPENDIX B

Cisco Unified TAPI Interfaces

This appendix contains a listing of APIs that are supported and not supported.

Cisco Unified TAPI Version 2.1 Interfaces

Core Package

The following table lists each TAPI interface

Table 145: Compliance to TAPI 2.1

API/Message/Structure	Cisco TAPI support	Comments
TAPI Line Functions		
lineAccept	Yes	
lineAddProvider	Yes	
lineAddToConference	Yes	
lineAnswer	Yes	
lineBlindTransfer	Yes	
lineCallbackFunc	Yes	
lineClose	Yes	
lineCompleteCall	No	
lineCompleteTransfer	Yes	
lineConfigDialog	No	
lineConfigDialogEdit	No	
lineConfigProvider	Yes	

API/Message/Structure	Cisco TAPI support	Comments
lineDeallocateCall	Yes	
lineDevSpecific	Yes	
lineDevSpecificFeature	Yes	
lineDial	Yes	
lineDrop	Yes	
lineForward	Yes	
lineGatherDigits	No	
lineGenerateDigits	Yes	
lineGenerateTone	Yes	
lineGetAddressCaps	Yes	
lineGetAddressID	Yes	
lineGetAddressStatus	Yes	
lineGetAppPriority	No	
lineGetCallInfo	Yes	
lineGetCallStatus	Yes	
lineGetConfRelatedCalls	Yes	
lineGetCountry	No	
lineGetDevCaps	Yes	
lineGetDevConfig	No	
lineGetIcon	No	
lineGetID	Yes	
lineGetLineDevStatus	Yes	
lineGetMessage	Yes	
lineGetNewCalls	Yes	
lineGetNumRings	Yes	
lineGetProviderList	Yes	
lineGetRequest	Yes	
lineGetStatusMessages	Yes	

API/Message/Structure	Cisco TAPI support	Comments
lineGetTranslateCaps	Yes	
lineHandoff	Yes	
lineHold	Yes	
lineInitialize	Yes	
lineInitializeEx	Yes	
lineMakeCall	Yes	
lineMonitorDigits	Yes	
lineMonitorMedia	No	
lineMonitorTones	Yes	
lineNegotiateAPIVersion	Yes	
lineNegotiateExtVersion	Yes	
lineOpen	Yes	
linePark	Yes	
linePickup	No	
linePrepareAddToConference	Yes	
lineRedirect	Yes	
lineRegisterRequestRecipient	Yes	
lineReleaseUserUserInfo	No	
lineRemoveFromConference	No	
lineRemoveProvider	Yes	
lineSecureCall	No	
lineSendUserUserInfo	No	
lineSetAppPriority	Yes	
lineSetAppSpecific	No	
lineSetCallData	No	
lineSetCallParams	No	
lineSetCallPrivilege	Yes	
lineSetCallQualityOfService	No	

API/Message/Structure	Cisco TAPI support	Comments
lineSetCallTreatment	No	
lineSetCurrentLocation	No	
lineSetDevConfig	No	
lineSetLineDevStatus	No	
lineSetMediaControl	No	
lineSetMediaMode	No	
lineSetNumRings	Yes	
lineSetStatusMessages	Yes	
lineSetTerminal	No	
lineSetTollList	Yes	
lineSetupConference	Yes	
lineSetupTransfer	Yes	
lineShutdown	Yes	
lineSwapHold	No	
lineTranslateAddress	Yes	
lineTranslateDialog	Yes	
lineUncompleteCall	No	
lineUnhold	Yes	
lineUnpark	Yes	
TAPI Line Messages		
LINE_ADDRESSSTATE	Yes	
LINE_APPNEWCALL	Yes	
LINE_CALLINFO	Yes	
LINE_CALLSTATE	Yes	
LINE_CLOSE	Yes	
LINE_CREATE	Yes	
LINE_DEVSPECIFIC	Yes	
LINE_DEVSPECIFICFEATURE	Yes	

API/Message/Structure	Cisco TAPI support	Comments
LINE_GATHERDIGITS	Yes	
LINE_GENERATE	Yes	
LINE_LINEDEVSTATE	Yes	
LINE_MONITORDIGITS	Yes	
LINE_MONITORMEDIA	No	
LINE_MONITORTONE	Yes	
LINE_REMOVE	Yes	
LINE_REPLY	Yes	
LINE_REQUEST	Yes	
TAPI Line Structures		
LINEADDRESSCAPS	Yes	
LINEADDRESSSTATUS	Yes	
LINEAPPINFO	Yes	
LINECALLINFO	Yes	
LINECALLLIST	Yes	
LINECALLPARAMS	Yes	
LINECALLSTATUS	Yes	
LINECALLTREATMENTENTRY	No	
LINECARDENTRY	Yes	
LINECOUNTRYENTRY	Yes	
LINECOUNTRYLIST	Yes	
LINEDEVCAPS	Yes	
LINEDEVSTATUS	Yes	
LINEDIALPARAMS	No	
LINEEXTENSIONID	Yes	
LINEFORWARD	Yes	
LINEFORWARDLIST	Yes	
LINEGENERATETONE	Yes	

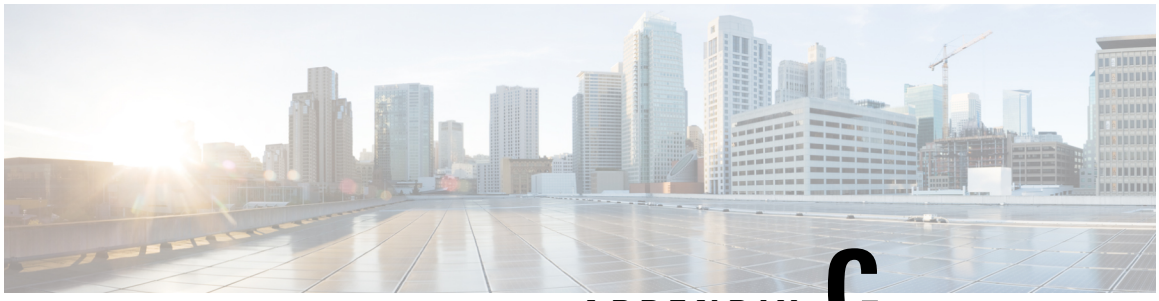
API/Message/Structure	Cisco TAPI support	Comments
LINEINITIALIZEEXPARAMS	Yes	
LINELOCATIONENTRY	Yes	
LINEMEDIACONTROLCALLSTATE	No	
LINEMEDIACONTROLDIGIT	No	
LINEMEDIACONTROLMEDIA	No	
LINEMEDIACONTROLTONE	No	
LINEMESSAGE	Yes	
LINEMONITORTONE	Yes	
LINEPROVIDERENTRY	Yes	
LINEPROVIDERLIST	Yes	
LINEREQMEDIACALL	No	
LINEREQMAKECALL	Yes	
LINETERMCAPS	No	
LINETRANSLATECAPS	Yes	
LINETRANSLATEOUTPUT	Yes	
TAPI Phone Functions		
phoneCallbackFunc	Yes	
phoneClose	Yes	
phoneConfigDialog	No	
phoneDevSpecific	Yes	
phoneGetButtonInfo	No	
phoneGetData	No	
phoneGetDevCaps	Yes	
phoneGetDisplay	Yes	
phoneGetGain	No	
phoneGetHookSwitch	No	
phoneGetIcon	No	
phoneGetID	No	

API/Message/Structure	Cisco TAPI support	Comments
phoneGetLamp	No	
phoneGetMessage	Yes	
phoneGetRing	Yes	
phoneGetStatus	No	
phoneGetStatusMessages	Yes	
phoneGetVolume	No	
phoneInitialize	Yes	
phoneInitializeEx	Yes	
phoneNegotiateAPIVersion	Yes	
phoneNegotiateExtVersion	No	
phoneOpen	Yes	
phoneSetButtonInfo	No	
phoneSetData	No	
phoneSetDisplay	Yes	
phoneSetGain	No	
phoneSetHookSwitch	No	
phoneSetLamp	No	
phoneSetRing	No	
phoneSetStatusMessages	Yes	
phoneSetVolume	No	
phoneShutdown	Yes	
TAPI Phone Messages		
PHONE_BUTTON	Yes	
PHONE_CLOSE	Yes	
PHONE_CREATE	Yes	
PHONE_DEVSPECIFIC	No	
PHONE_REMOVE	Yes	
PHONE_REPLY	Yes	

API/Message/Structure	Cisco TAPI support	Comments
PHONE_STATE	Yes	
TAPI Phone Structures		
PHONEBUTTONINFO	No	
PHONECAPS	Yes	
PHONEEXTENSIONID	No	
PHONEINITIALIZEEXPARAMS	Yes	
PHONEMESSAGE	Yes	
PHONESTATUS	No	
VARSTRING	Yes	
TAPI Assisted Telephony Functions		
tapiRequestDrop	No	
tapiRequestMediaCall	No	
TAPI Call Center Functions		
lineAgentSpecific	No	
lineGetAgentActivityList	No	
lineGetAgentCaps	No	
lineGetAgentGroupList	No	
lineGetAgentStatus	No	
lineProxyMessage	No	
lineProxyResponse	No	
lineSetAgentActivity	No	
lineSetAgentGroup	No	
lineSetAgentState	No	
TAPI Call Center Messages		
LINE_AGENTSPECIFIC	No	
LINE_AGENTSTATUS	No	
LINE_PROXYREQUEST	No	
TAPI Call Center Structures		

API/Message/Structure	Cisco TAPI support	Comments
LINEAGENTACTIVITYENTRY	No	
LINEAGENTACTIVITYLIST	No	
LINEAGENTCAPS	No	
LINEAGENTGROUPENTRY	No	
LINEAGENTGROUPLIST	No	
LINEAGENTSTATUS	No	
LINEPROXYREQUEST	No	
Wave Functions		
waveInAddBuffer	Yes	
waveInClose	Yes	
waveInGetDevCaps	No	
waveInGetErrorText	No	
waveInGetID	Yes	
waveInGetNumDevs	No	
waveInGetPosition	Yes	
waveInMessage	No	
waveInOpen	Yes	
waveInPrepareHeader	Yes	
waveInProc	No	
waveInReset	Yes	
waveInStart	Yes	
waveInStop	No	
waveInUnprepareHeader	Yes	
waveOutBreakLoop	No	
waveOutClose	Yes	
waveOutGetDevCaps	Yes	
waveOutGetErrorText	No	
waveOutGetID	Yes	

API/Message/Structure	Cisco TAPI support	Comments
waveOutGetNumDevs	No	
waveOutGetPitch	No	
waveOutGetPlaybackRate	No	
waveOutGetPosition	No	
waveOutGetVolume	No	
waveOutMessage	No	
waveOutOpen	Yes	
waveOutPause	No	
waveOutPrepareHeader	Yes	
waveOutProc	No	
waveOutReset	Yes	
waveOutRestart	No	
waveOutSetPitch	No	
waveOutSetPlaybackRate	No	
waveOutSetVolume	No	
waveOutUnprepareHeader	Yes	
waveOutWrite	Yes	



APPENDIX C

Troubleshooting Cisco Unified TAPI

This appendix contains information about troubleshooting Cisco Unified Communication manager. It contains the following sections:

TSP Trace of Internal Messages

Procedure

Step 1 Choose **Start > Settings > Control Panel** and select **Phone and Modem Options**.

Step 2 Click **Advanced** tab and select the **CiscoTSP 0xx** and click **Configure** button.

Step 3 Click **Trace** tab. Select **Trace On** check box and select **1**. **TSP Trace** to trace the TSP internal messages. Select **Error** to just log errors in the TSP Select **Detailed** to log internal messages for debugging purposes. Select **2. CTI Trace** to trace the messages sent between CTI and TSP. Select **3. TSPI Trace** to trace the requests and events that are sent between TSP and TAPI.

Step 4 Set up a Directory that is the path for the trace log. For example, c:\Temp No. of Files: Setting this to a value greater than or equal to 1 enables rolling log files. For example, a value of 10 will cause up to 10 log files to be used in a cyclic fashion. Max lines/file: specifies the maximum number of trace statements that will be written to each log file. For example, a value of 1000 will cause up to 1000 trace statements to be written to each log file.

TSP Operation Verification

To verify the TSP operation on the machine where the TSP is installed, use the Microsoft Windows Phone Dialer Application. Find this application in the C:\Program Files\Windows NT directory under the name dialer.exe. When the program is run, a dialog box displays that asks which line and address the user wants to use to connect. If there are no lines in the Line drop down list, then a problem may exist between the TSP and the Cisco Unified Communications Manager. If lines are available, choose one of the lines, keep the Address

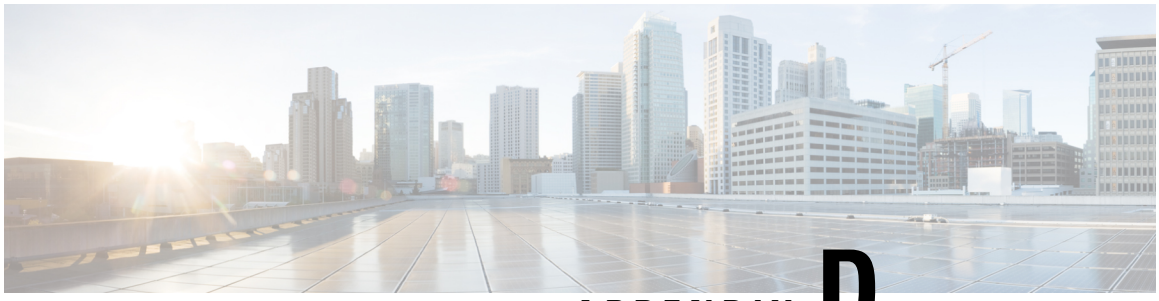
set to zero (0) and click **OK**. Enter a Number to dial, and a call should be placed to that number. If call is successful, you know that the TSP is operational on the machine where the TSP is installed. If problems are encountered with installation and setup of Remote TSP, this test represents a good way to verify whether the TSP is operating properly and that the problem is with the configuration and setup of Remote TSP.

Version Compatibility

Cisco recommends that the TSP client should always use the plug-in that is downloaded from corresponding Cisco Unified Communications Manager server.

Cisco TSP Readme

The Cisco Unified Communications Manager TSP readme file is copied to the client PC when TSP plug-in is installed.



APPENDIX D

Cisco Unified TAPI Operations-by-Release

Cisco Unified TAPI Operations-by-Release

The following tables list new, changed, and "under consideration or review" features for Cisco Unified TAPI by Cisco Unified Communications Manager release.

- API Interfaces
- TAPI Line Functions
- TAPI Line Messages
- TAPI Line Structures
- TAPI Phone Functions
- TAPI Phone Messages
- TAPI Phone Structures

Table legend:

s: supported, N: not supported, M: Modified, UCR: Under Consideration or Review.

Table 146: API Interfaces

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
CTI Manager and Support for fault tolerance	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for Cisco CallManager Extension Mobility	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for Multiple CiscoTSP	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

Cisco Unified TAPI Operations-by-Release

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
(Redirect Support for) Blind Transfer	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for swap hold and setup transfer with the lineDevSpecific() function	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for lineForward()	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support to Reset the Original Called Party upon Redirect with the lineDevSpecific function	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support to Set the Original Called Party upon Redirect with the lineDevSpecific function	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Line In-Service or Out-of-Service	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for multiple languages in the CiscoTSP installation program and in the CiscoTSP configuration dialogs	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
User Deletion from Directory	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Opening Two Lines on One CTI Port Device	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for linePark and lineUnpark	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for monitoring Call Park Directory Numbers using lineOpen	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Call Reason Enhancements	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Device Data Passthrough	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
CiscoTSP Auto Update	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

Cisco Unified TAPI Operations-by-Release

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
Multiple Calls per Line Appearance	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Shared Line Appearance	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Select Calls	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Transfer Changes	N	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Direct Transfer	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Conference Changes	N	N	N	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Join	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Privacy Release	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Barge and cBarge	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Dynamic Port Registration	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Media Termination at Route Points	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
QoS support	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Support for Presentation Indication	N	N	N	s	s	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s
Windows 2003 Support	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Unicode Support	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s
SRTP support	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Partition Support	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SuperProvider Functionality	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Security (TLS) support	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
FAC/CMC Support	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
CTI Port Third Party Monitoring	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Alternate Script Support	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SIP Features Refer/Replaces	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
SIP URI	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s

Cisco Unified TAPI Operations-by-Release

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
SIP phone support	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Change Notification of SupetProvider and CallParkDN Monitoring flags	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
3XX	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s
Intercom Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
Secure Conferencing Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
Monitoring & Recording	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
Arabic and Hebrew Language Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
Do-Not-Disturb Support	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
Conference Enhancement	N	N	N	N	N	s	s	N	N	s	s	s	s	s	s	s	s	s	s	s
Join AcrossLine (SCCP)	N	N	N	N	N	N	N	s	s	N	s	s	s	s	s	s	s	s	s	s
Join AcrossLine (SIP)	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Locale Infrastructure Enhancement	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Do-Not-Disturb Rejection	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Call Party Normalization	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
Click-To-Conference	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s
IPv6 Support on Linux	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Windows Vista Support	N	N	N	N	N	s	s	N	s	N	s	s	s	s	s	s	s	s	s	s
Enhanced MWI	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Direct Transfer Across Lines	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Support for > 100DNs	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Swap/Cancel support on Cisco Unified IP Phone 8900 and 9900 Series	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Drop Any Party	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
Park Reversion	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Conditional Reset	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Logical Partition	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Assisted DPark	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Cisco Unified IP Phone 6900 Series Support	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Device State Server	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s
Exposing Busy Trigger / Line Number / Voice Mail Pilot / Line Label / New call outbound rollover/ Consult call rollover/JAL/DTAL flag and IP address (IPv4 & IPv6) of the device	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s
Hunt List Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Call Intercept Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
External Call Control (ECC) Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Call Control Discover (CCD) Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Extension Mobility Cross Cluster (EMCC) Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Call Pickup Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
End-To-End Call Tracing	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Secure Monitoring Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Unified B2B link support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
iSAC Codec Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
New TSP Client with remote silent installation	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s

Cisco Unified TAPI Operations-by-Release

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
New Cisco TSP Wave Driver (Cisco RTP Library)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s
Agent Greeting	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Agent Zip Tone	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Early Offer	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Extension Mobility Memory Optimization	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Other-Device State Notification	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
Energy Wise	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Whisper Coaching	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
FIPS Compliant (UCR 2008 support)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Password Expiry and Account Lockout (UCR 2008 support)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Support for Codian SIP MCU	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
TSP Native 64Bit support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s
TSP Native 64Bit SRTP support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Support for multiple calls per line on RTLite Phones	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s
Single Sign On	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
URI Dialing	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
Recording Enhancement (Device Based Recording)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
Hunt Pilot Connected Number	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
Native Queuing	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
CIUS Session Persistency	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s

TSP Features	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	8.0	8.5	8.6	9.0	9.1	10.0
CTI Remote Device (Cisco Extend & Connect)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s
CTI Remote Device - Extend Mode for CSF Removed	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
CTI Remote Device - ADR (Application Dial Rule) Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
CTI Remote Device - Remote Destination Reachability Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
CTI Remote Device - DTMF Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s	s
CTI Remote Device -Persistent Call	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CTI Remote Device -Announcement Call	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CTI Remote Device -Call Forwarding	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CTI Remote Device - NuRD	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CTI Remote Device - Mobility Interaction Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CTI Video Support	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
Gateway Recording	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
CCMSymmetric Encryption Enhancements -AsymmetricEncryption	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s
Video on Hold	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	s

Table 147: TAPI Line Functions

TAPI Line Functions	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	7.1.3	8.0	8.5	8.6	9.0	9.1	10.0
LineAddToConference	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

TAPI Line Functions	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	7.1.3	8.0	8.5	8.6	9.0	9.1	10.0
LineCompleteTransfer	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineDevSpecific	M	s	s	M	M	s	s	s	M	M	s	M	M	s	M	M	M	M	M	s
LineForward	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineMakeCall	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LinePark	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineUnpark	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineRedirect	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineBlindTransfer	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LineDevSpecificFeature	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
LineRemoveFromConference	N	N	N	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s

Table 148: TAPI Line Messages

TAPI Line Messages	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	7.1.3	8.0	8.5	8.6	9.0	9.1	10.0
LINE_ADDRESSSTATE	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LINE_CALLINFO	M	s	s	s	s	s	s	M	M	M	M	M	M	s	M	M	M	s	s	s
LINE_CALLSTATE	s	s	s	M	M	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LINE_REMOVE	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
LINE_DEVSPECIFIC	s	s	s	s	M	s	s	s	M	M	s	s	s	M	M	s	M	M	s	M
LINE_DEVSPECIFICFEATURE	N	N	N	N	N	N	N	N	N	s	s	s	s	s	s	s	s	s	s	s
LINE_CALLDEVSPECIFIC	s	s	s	s	M	s	s	s	M	M	s	s	M	s	M	M	s	s	s	M

Table 149: TAPI Line Structures

TAPI Line Structures	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	8.0	8.5	8.6	9.0	10.0
LINEADDRESSCAPS	M	s	s	M	M	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LINECALLSTATUS	s	s	s	M	M	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LINEFORWARD	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LINEFORWARDLIST	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M
LINEDEVCAPS	s	s	M	s	s	s	s	M	M	M	s	s	M	M	M	s	s	M	M

TAPI Line Structures	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	8.0	8.5	8.6	9.0	10.0
LINEDEVSTATUS	s	s	s	s	s	s	s	M	s	M	s	s	s	s	s	s	s	s	s
LINECALLINFO	s	s	s	s	s	s	s	s	M	M	M	M	M	M	M	M	s	s	M
LINECALLPARAMs	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M

Table 150: TAPI Phone Functions

TAPI Phone Functions	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	8.0	8.5	8.6	9.0	10.0
PhoneDevSpecific	N	N	s	s	s	s	s	M	s	s	s	s	s	s	s	s	s	s	s
PhoneGetStatus	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

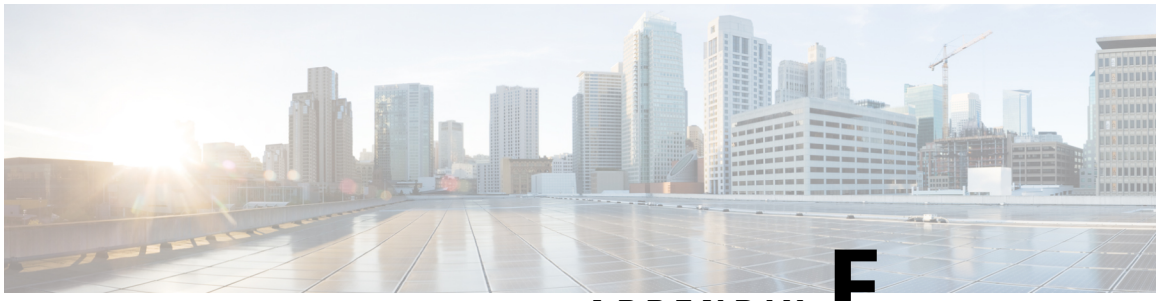
Table 151: TAPI Phone Messages

TAPI Phone Messages	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	5.2	6.0	6.1	7.0	7.1.2	7.1.3	8.0	8.5	8.6	9.0	10.0
PHONE_REMOVE	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
PHONE_DEVSPECIFIC	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M	M	s	s	s	M

Table 152: TAPI Phone Structures

TAPI Phone Structures	3.1	3.2	3.3	4.0	4.1	4.2	4.3	5.0	5.1	5.2	6.0	6.1	7.0	7.1.2	7.1.3	8.0	8.5	8.6	9.0	10.0
PHONECAPS	s	s	s	s	s	s	s	s	s	s	s	s	s	s	M	M	s	s	s	M
PHONESTATUS	N	N	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

Cisco Unified TAPI Operations-by-Release



APPENDIX E

CTI Supported Devices

CTI Supported Devices

The following table provides information about CTI supported devices.

Table legend:

s: supported, N: not supported, NA: Not Applicable.

Table 153: CTI Supported Device Matrix

Device/Phone model	SCCP	SIP	Comments
Analog Phone	s	N	Limitations for this device described in the JTAPI Developers Guide 4.1(3)
Cisco 12 SP	s	N	End of Software Maintenance Release 2001
Cisco 30 SP+	s	N	End of Software Maintenance Release 2001
Cisco Unified IP Phone 3911	NA	NA	Not a CTI supported device
Cisco Unified IP Phone 6901	s	s	SIP requires firmware 9.1(1)
Cisco Unified IP Phone 6911	s	s	SIP requires firmware 9.1(1)
Cisco Unified IP Phone 6921	s	s	phoneSetDisplay() interface is not supported. SIP requires firmware 9.1(1)
Cisco Unified IP Phone 6941	s	s	phoneSetDisplay() interface is not supported. SIP requires firmware 9.1(1)

Device/Phone model	SCCP	SIP	Comments
Cisco Unified IP Phone 6945	s	s	phoneSetDisplay() interface is not supported. SIP requires firmware 9.1(1)
Cisco Unified IP Phone 6961	s	s	phoneSetDisplay() interface is not supported. SIP requires firmware 9.1(1)
Cisco Unified IP Phone 7821	s	s	
Cisco Unified IP Phone 7841	s	s	
Cisco Unified IP Phone 7902	s	N	End of Software Maintenance Release 2007
Cisco Unified IP Phone 7905	s	N	End of Software Maintenance Release 2007
Cisco Unified IP Phone 7906	s	s	
Cisco Unified IP Phone 7910	s	N	End of Software Maintenance Release 2007
Cisco Unified IP Phone 7911	s	s	
Cisco Unified IP Phone 7912	s	N	End of Software Maintenance Release 2007
Cisco Unified IP Phone 7914 Sidecar	s	N	End of Software Maintenance Release 2010
Cisco Unified IP Phone 7915 Sidecar	s	s	
Cisco Unified IP Phone 7916 Sidecar	s	s	
Cisco Unified IP Phone 7920	s	N	End of Software Maintenance Release 2007
Cisco Unified IP Phone 7921	s	N	
Cisco Unified IP Phone 7925 and 7925-EX	s	N	
Cisco Unified IP Phone 7931	s	N	CTI supported only if rollover is disabled. Starting 7.1 this device is supported when corresponding role is added to user.
Cisco Unified IP Phone 7935	s	N	End of Software Maintenance Release 2007

Device/Phone model	SCCP	SIP	Comments
Cisco Unified IP Phone 7936	s	N	End of Software Maintenance Release 2011
Cisco Unified IP Phone 7937	s	N	
Cisco Unified IP Phone 7940	s	N	End of Software Maintenance Release 2011
Cisco Unified IP Phone 7941	s	s	
Cisco Unified IP Phone 7941G-GE	s	s	End of Software Maintenance Release 2009
Cisco Unified IP Phone 7942	s	s	
Cisco Unified IP Phone 7945	s	s	
Cisco Unified IP Phone 7960	s	N	End of Software Maintenance Release 2011
Cisco Unified IP Phone 7961	s	s	
Cisco Unified IP Phone 7961G-GE	s	s	End of Software Maintenance Release 2009
Cisco Unified IP Phone 7962	s	s	
Cisco Unified IP Phone 7965	s	s	
Cisco Unified IP Phone 7970	s	s	End of Software Maintenance Release 2009
Cisco Unified IP Phone 7971	s	s	End of Software Maintenance Release 2009
Cisco Unified IP Phone 7975	s	s	
Cisco Unified IP Phone 7985	s	N	End of Software Maintenance Release 2011
Cisco Unified IP Phone 8811	N	s	8811 phones are CTI controlled
Cisco Unified IP Phone 8941	s	N	
Cisco Unified IP Phone 8945	s	N	
Cisco Unified IP Phone 8961	N	s	phoneSetDisplay() interface is not supported
Cisco Unified IP Phone 9951	N	s	phoneSetDisplay() interface is not supported
Cisco Unified IP Phone 9971	N	s	phoneSetDisplay() interface is not supported

Device/Phone model	SCCP	SIP	Comments
Cisco ATA 186	s	N	For details on the limitations of this device, see
Cisco Cius	N	s	CTI support added in 8.5(1). phoneSetDisplay() interface is not supported. XSI interface is not supported. Silent Monitoring/Recording is not supported.
Cisco IP Communicator	s	s	
Cisco Unified Personal Communicator -Softphone Mode	s	s	CTI support added in 8.5(1)
Cisco Unified Personal Communicator -Remote Desktop Control Mode	NA	NA	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Unified Communications Integration for Microsoft Office Communicator/Lync -Softphone Mode	N	s	CTI support added in 8.5(2)
Cisco Unified Communications Integration for Microsoft Office Communicator/Lync -Remote Desktop Control Mode	NA	NA	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Web Communicator for Quad -Softphone Mode	NA	NA	Not a CTI supported device
Cisco Web Communicator for Quad -Remote Desktop Control Mode	NA	NA	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco Unified Communications Integration for WebEx Connect -Softphone Mode	NA	NA	Not a CTI supported device

Device/Phone model	SCCP	SIP	Comments
Cisco Unified Communications Integration for WebEx Connect -Remote Desktop Control Mode	NA	NA	Refer to the device model under remote control to determine CTI support. Click-to-Answer requires device speakerphone support.
Cisco VGC Phone	s	N	
VG224	NA	NA	Not a CTI supported device
VG248	s	N	For details on the limitations of this device, see
CTI Port	NA	NA	CTI supported virtual device
CTI Route Point	NA	NA	CTI supported virtual device
CTI Route Point (Pilot)	NA	NA	CTI supported virtual device
ISDN BRI Phone	NA	NA	Not a CTI supported device

CTI Supported Devices