



Cisco Unified Communications Manager Express Telephony Service Provider 2.1 Developer's Guide

January 2007

Corporate 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: 408 526-4100



THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCSP, the Cisco Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Access Registrar, Aironet, ASIST, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Empowering the Internet Generation, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, Packet, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, ScriptShare, SlideCast, SMARTnet, StrataView Plus, SwitchProbe, TeleRouter, The Fastest Way to Increase Your Internet Quotient, TransPath, and VCO are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0501R)

Cisco Unified Communications Manager Express Telephony Service Provider 2.1 Developer's Guide
© 2007 Cisco Systems, Inc. All rights reserved.



Preface	vii
Obtaining Documentation	vii
Cisco.com	vii
Product Documentation DVD	vii
Ordering Documentation	viii
Documentation Feedback	viii
Cisco Product Security Overview	viii
Reporting Security Problems in Cisco Products	ix
Obtaining Technical Assistance	ix
Cisco Technical Support & Documentation Website	x
Submitting a Service Request	x
Definitions of Service Request Severity	xi
Obtaining Additional Publications and Information	xi
Introduction	1
Purpose	2
Audience	2
Organization	2
Related Documents	2
Required Software	3
Supported Windows Platforms	3
Terminology	3
Overview	5
Call Control	7
First-Party Call Control	7
Third-Party Call Control	7
Multiple TAPI Applications	7
Compatibility	8
Supported Device and Line Types	8
Startup with Windows	9
Resets and Restarts	9
Debug Tracing	9
Exception Notes	10

Call Origin	10
Line Type	10
Shared Dual Lines	10
Outgoing Calls on Shared Lines	10
MAC Address/Device ID	10
Cisco Unified CME TAPI Line Device	11
Cisco Unified CME TAPI Line Functions	11
lineAddtoConference	11
lineAnswer	12
lineBlindTransfer	13
lineCallbackFunc	14
lineClose	15
lineCompleteCall	16
lineCompleteTransfer	17
lineConfigProvider	19
lineDeallocateCall	19
lineDevSpecificFeature	20
lineDial	21
lineDrop	22
lineForward	23
lineGetAddressCaps	25
lineGetAddressStatus	27
lineGetCallInfo	28
lineGetCallStatus	28
lineGetDevCaps	29
lineGetID	30
lineHold	31
lineInitializeEx	32
lineMakeCall	33
lineNegotiateAPIVersion	34
lineOpen	35
linePark	37
linePickup	38
lineRemoveProvider	39
lineSetupConference	40
lineSetupTransfer	41
lineUnhold	42
lineUnpark	42
Cisco Unified CME TAPI Line Messages	44

LINE_ADDRESSTATE	44
LINE_APPNEWCALL	45
LINE_CALLINFO	46
LINE_CALLSTATE	47
LINE_CLOSE	49
LINE_CREATE	50
LINE_DEVSPECIFIC	51
LINE_DEVSTATE	52
LINE_REMOVE	53
LINE_REPLY	53
Cisco Unified CME TAPI Line Structures	55
LINEADDRESSCAPS	55
LINEADDRESSSTATUS	61
LINEAAPPINFO	62
LINECALLINFO	63
LINECALLLIST	70
LINECALLPARMS	71
LINECALLSTATUS	76
LINEDEVCAPS	77
LINEDEVSTATUS	83
LINEFORWARD	85
LINEMESSAGE	86
Cisco Unified CME TAPI Phone Device	87
Cisco Unified CME TAPI Phone Functions	88
phoneCallbackFunc	88
phoneClose	89
phoneGetDevCaps	90
phoneGetDisplay	91
phoneGetHookswitch	91
phoneGetMessage	92
phoneGetRing	93
phoneInitializeEx	93
phoneNegotiateAPIVersion	95
phoneOpen	96
phoneSetHookswitch	97
phoneSetRing	98
phoneSetVolume	99
phoneShutdown	100
Cisco Unified CME TAPI Phone Messages	101

- PHONE_CLOSE 101
- PHONE_REMOVE 102
- PHONE_REPLY 103
- Cisco Unified CME TAPI Phone Structures 104
 - PHONECAPS 104
 - PHONEEXTENSIONID 109
 - PHONEMESSAGE 109

Index



Preface

Revised: January 12, 2007

Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation at this URL:

<http://www.cisco.com/techsupport>

You can access the Cisco website at this URL:

<http://www.cisco.com>

You can access international Cisco websites at this URL:

http://www.cisco.com/public/countries_languages.shtml

Product Documentation DVD

The Product Documentation DVD is a comprehensive library of technical product documentation on a portable medium. The DVD enables you to access multiple versions of installation, configuration, and command guides for Cisco hardware and software products. With the DVD, you have access to the same HTML documentation that is found on the Cisco website without being connected to the Internet. Certain products also have .PDF versions of the documentation available.

The Product Documentation DVD is available as a single unit or as a subscription. Registered Cisco.com users (Cisco direct customers) can order a Product Documentation DVD (product number DOC-DOCDVD= or DOC-DOCDVD=SUB) from Cisco Marketplace at this URL:

<http://www.cisco.com/go/marketplace/>

Ordering Documentation

Registered Cisco.com users may order Cisco documentation at the Product Documentation Store in the Cisco Marketplace at this URL:

<http://www.cisco.com/go/marketplace/>

Nonregistered Cisco.com users can order technical documentation from 8:00 a.m. to 5:00 p.m. (0800 to 1700) PDT by calling 1 866 463-3487 in the United States and Canada, or elsewhere by calling 011 408 519-5055. You can also order documentation by e-mail at tech-doc-store-mkpl@external.cisco.com or by fax at 1 408 519-5001 in the United States and Canada, or elsewhere at 011 408 519-5001.

Documentation Feedback

You can rate and provide feedback about Cisco technical documents by completing the online feedback form that appears with the technical documents on Cisco.com.

You can submit comments about Cisco documentation by using the response card (if present) behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Customer Document Ordering
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Cisco Product Security Overview

Cisco provides a free online Security Vulnerability Policy portal at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

From this site, you will find information about how to:

- Report security vulnerabilities in Cisco products.
- Obtain assistance with security incidents that involve Cisco products.
- Register to receive security information from Cisco.

A current list of security advisories, security notices, and security responses for Cisco products is available at this URL:

<http://www.cisco.com/go/psirt>

To see security advisories, security notices, and security responses as they are updated in real time, you can subscribe to the Product Security Incident Response Team Really Simple Syndication (PSIRT RSS) feed. Information about how to subscribe to the PSIRT RSS feed is found at this URL:

http://www.cisco.com/en/US/products/products_psirt_rss_feed.html

Reporting Security Problems in Cisco Products

Cisco is committed to delivering secure products. We test our products internally before we release them, and we strive to correct all vulnerabilities quickly. If you think that you have identified a vulnerability in a Cisco product, contact PSIRT:

- For Emergencies only — security-alert@cisco.com

An emergency is either a condition in which a system is under active attack or a condition for which a severe and urgent security vulnerability should be reported. All other conditions are considered nonemergencies.

- For Nonemergencies — psirt@cisco.com

In an emergency, you can also reach PSIRT by telephone:

- 1 877 228-7302
- 1 408 525-6532



Tip

We encourage you to use Pretty Good Privacy (PGP) or a compatible product (for example, GnuPG) to encrypt any sensitive information that you send to Cisco. PSIRT can work with information that has been encrypted with PGP versions 2.x through 9.x.

Never use a revoked or an expired encryption key. The correct public key to use in your correspondence with PSIRT is the one linked in the Contact Summary section of the Security Vulnerability Policy page at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

The link on this page has the current PGP key ID in use.

If you do not have or use PGP, contact PSIRT at the aforementioned e-mail addresses or phone numbers before sending any sensitive material to find other means of encrypting the data.

Obtaining Technical Assistance

Cisco Technical Support provides 24-hour-a-day award-winning technical assistance. The Cisco Technical Support & Documentation website on Cisco.com features extensive online support resources. In addition, if you have a valid Cisco service contract, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not have a valid Cisco service contract, contact your reseller.

Cisco Technical Support & Documentation Website

The Cisco Technical Support & Documentation website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day, at this URL:

<http://www.cisco.com/techsupport>

Access to all tools on the Cisco Technical Support & Documentation website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

<http://tools.cisco.com/RPF/register/register.do>

**Note**

Use the Cisco Product Identification (CPI) tool to locate your product serial number before submitting a web or phone request for service. You can access the CPI tool from the Cisco Technical Support & Documentation website by clicking the **Tools & Resources** link under Documentation & Tools. Choose **Cisco Product Identification Tool** from the Alphabetical Index drop-down list, or click the **Cisco Product Identification Tool** link under Alerts & RMAs. The CPI tool offers three search options: by product ID or model name; by tree view; or for certain products, by copying and pasting **show** command output. Search results show an illustration of your product with the serial number label location highlighted. Locate the serial number label on your product and record the information before placing a service call.

Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool provides recommended solutions. If your issue is not resolved using the recommended resources, your service request is assigned to a Cisco engineer. The TAC Service Request Tool is located at this URL:

<http://www.cisco.com/techsupport/servicerequest>

For S1 or S2 service requests, or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)

EMEA: +32 2 704 55 55

USA: 1 800 553-2447

For a complete list of Cisco TAC contacts, go to this URL:

<http://www.cisco.com/techsupport/contacts>

Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

Severity 1 (S1)—An existing network is down, or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Severity 2 (S2)—Operation of an existing network is severely degraded, or significant aspects of your business operations are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Severity 3 (S3)—Operational performance of the network is impaired, while most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- The *Cisco Product Quick Reference Guide* is a handy, compact reference tool that includes brief product overviews, key features, sample part numbers, and abbreviated technical specifications for many Cisco products that are sold through channel partners. It is updated twice a year and includes the latest Cisco offerings. To order and find out more about the Cisco Product Quick Reference Guide, go to this URL:

<http://www.cisco.com/go/guide>

- Cisco Marketplace provides a variety of Cisco books, reference guides, documentation, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

<http://www.cisco.com/go/marketplace/>

- *Cisco Press* publishes a wide range of general networking, training and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:

<http://www.ciscopress.com>

- *Packet* magazine is the Cisco Systems technical user magazine for maximizing Internet and networking investments. Each quarter, Packet delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can access Packet magazine at this URL:

<http://www.cisco.com/packet>

- *iQ Magazine* is the quarterly publication from Cisco Systems designed to help growing companies learn how they can use technology to increase revenue, streamline their business, and expand services. The publication identifies the challenges facing these companies and the technologies to help solve them, using real-world case studies and business strategies to help readers make sound technology investment decisions. You can access iQ Magazine at this URL:

<http://www.cisco.com/go/iqmagazine>

or view the digital edition at this URL:

<http://ciscoiq.texterity.com/ciscoiq/sample/>

- *Internet Protocol Journal* is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

<http://www.cisco.com/ipj>

- Networking products offered by Cisco Systems, as well as customer support services, can be obtained at this URL:

<http://www.cisco.com/en/US/products/index.html>

- Networking Professionals Connection is an interactive website for networking professionals to share questions, suggestions, and information about networking products and technologies with Cisco experts and other networking professionals. Join a discussion at this URL:

<http://www.cisco.com/discuss/networking>

- World-class networking training is available from Cisco. You can view current offerings at this URL:

<http://www.cisco.com/en/US/learning/index.html>



Introduction

Last updated: November 14, 2008

This chapter introduces the Cisco Unified Communications Manager Express (Cisco Unified CME, formerly known as Cisco Unified CallManager Express) Telephony Application Programming Interface (TAPI) implementation, describes the purpose of this document, and outlines the required software. The chapter includes the following sections:

- [Purpose, page 2](#)
- [Audience, page 2](#)
- [Organization, page 2](#)
- [Related Documents, page 2](#)
- [Required Software, page 3](#)
- [Supported Windows Platforms, page 3](#)
- [Terminology, page 3](#)

TAPI comprises the set of classes and principles of operation that constitute a telephony application programming interface. TAPI implementations provide the interface between computer telephony applications and telephony services. Cisco Unified CME provides a telephony service provider (Cisco Unified CME TSP 2.1). Cisco Unified CME TSP 2.1 allows developers to create customized IP telephony applications for Cisco Unified CME users; for example, voice messaging with other TAPI-compliant systems, automatic call distribution (ACD), and caller ID screen popups.

Cisco Unified CME TSP 2.1 implementation uses the Microsoft TAPI v2.2 specification and supplies extension functions to support Cisco Unified IP Telephony Solutions. To enable a Cisco Unified CME TSP-based solution, you must have the following:

- TAPI support/service that is running on your Windows system
- A TAPI-based software application
- A Cisco Unified CME IP telephone system



Note

The system does not support using Cisco Unified CME TSP 2.1 via the TAPI 3.x compatibility layer.



Americas Headquarters:
Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

Purpose

This document describes the Cisco Unified CME 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 CME IP telephony hardware, software, and processes. A primary goal of a standard application programming interface (API), such as TAPI, is to provide an unchanging programming interface under which varied implementations may stand. Cisco's goal in implementing TAPI for the Cisco Unified CME 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 CME to applications.

Audience

Cisco intends this document to be for use by telephony software engineers who are developing Cisco Unified 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.

Organization

The organization of this manual is described in [Table 1](#).

Table 1 *Organization of Document*

Chapter	Description
Introduction	General information regarding target audience for the guide and sources of support.
Overview	Outlines the key concepts and describes changes in and enhancements to Cisco Unified CME TSP 2.1.
Cisco Unified CME TAPI Line Device	Describes the supported line device functions, messages and structures in the Cisco implementation of the standard Microsoft TAPI.
Cisco Unified CME TAPI Phone Device	Describes the supported phone device functions, messages and structures in the Cisco implementation of the standard Microsoft TAPI.

Related Documents

The following resources provide more information about TAPI specifications, creating an application to use TAPI, and TAPI administration:

- *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 who is a COMM and TAPI specialist*
- *TAPI 2.1 Microsoft TAPI Client Management*
- *TAPI 2.1 Administration Tool*

Required Software

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

Cisco Unified CME TSP 2.1 requires the following software:

- Cisco Unified Communications Manager Express version 12.3.11, 12.4

Supported Windows Platforms

All Windows operating systems support Cisco Unified CME TSP 2.1. Depending on the type and version of your operating system, you may need to install a service pack.

- Windows 2000
- Windows XP



Note

Check %SystemRoot%\system32 for these dynamically loaded library (.dll) files and versions:

- msvcrt.dll version: 6.00.8397.0
- msvcp60.dll version: 6.00.8168.0
- mfc42.dll version: 6.00.8447.0

Terminology

The terms shown in [Table 2](#) are used frequently in the manual to identify different kinds of individuals and objects.

Table 2 *Terms*

Term	Meaning
Administrator	The person responsible for the administration of the InstaRoute CallCenter system.
Agent	Person who answers ACD calls using an agent instrument.
Database	A database is a file that contains information in a tabular format.
Dialog	Popup window from which options are selected.
Directory	A directory is a database that usually contains names and related information.
Field	Each column of the database table is called a field.
Group	A supervisory collection of agents. A group is a division of a serving team that provides a supervisor with an easily manageable set of agents. Agents in the same group handle similar call types.
Queue	Displays a list of call queues. Queue is a first-in, first-out ranking of calls of one type waiting for agents to answer them. One group or serving team may service multiple call queues.

Table 2 *Terms (continued)*

Term	Meaning
Record or Entry	Each row of the database table is called an entry or a record.
Reports	A report is a pre-defined template or style for printing the information from a database.
Server	Named directory containing control and data files.
Supervisor	The person responsible for a group of agents.
Users	Individuals who are authorized to use the system.
Window	An area of the screen where the application displays information.



Overview

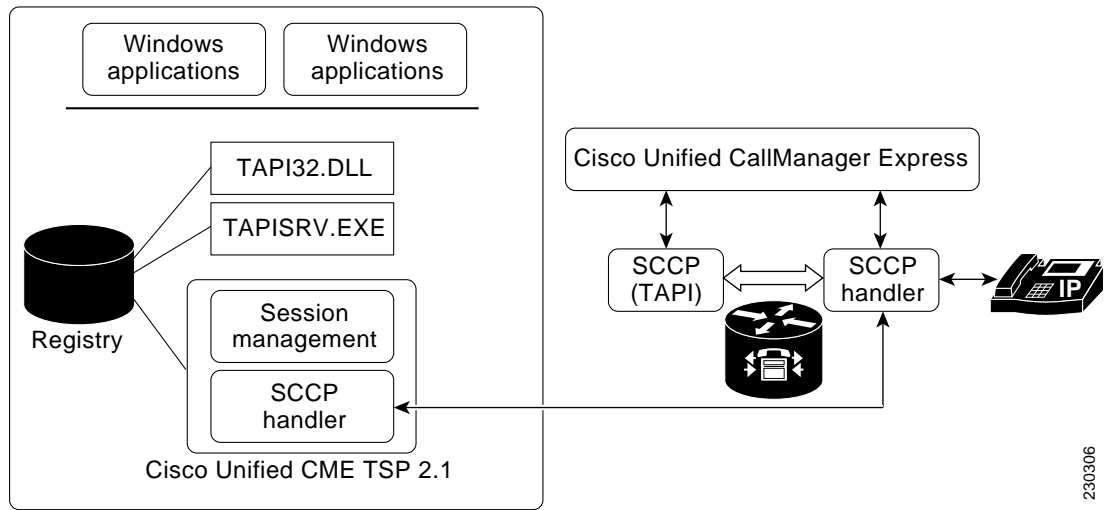
Revised: January 12, 2007

This chapter outlines the key concepts that are involved in using Cisco Unified CME TSP 2.1 and provides notes on the operation and implementation of Cisco Unified CME TSP 2.1.

- [Call Control, page 7](#)
- [Multiple TAPI Applications, page 7](#)
- [Compatibility, page 8](#)
- [Supported Device and Line Types, page 8](#)
- [Startup with Windows, page 9](#)
- [Resets and Restarts, page 9](#)
- [Debug Tracing, page 9](#)
- [Exception Notes, page 10](#)

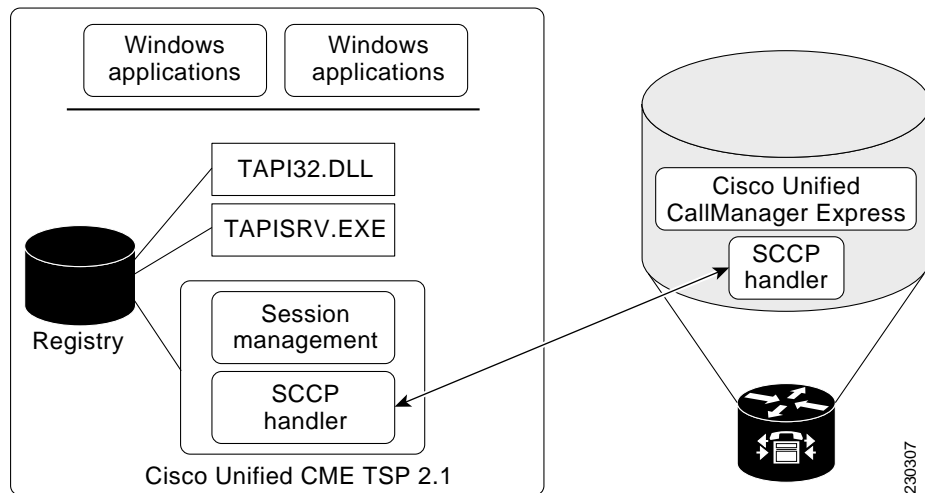
Cisco Unified CME TSP 2.1 connects to Cisco Unified CME endpoints and exposes the TAPI version 2.2 functions to Windows applications. Cisco Unified CME TSP 2.1 supports a single line device; that is, it can control or emulate only one ephone. [Figure 1](#) shows how various Cisco components fit into the Microsoft Windows telephony services in control mode of operation. [Figure 2](#) shows the similar components when the TSP is configured to terminate media in emulation mode.

Figure 1 *Windows Telephony Architecture and Cisco Unified CME TAPI Components in Control Mode*



Cisco Unified CME TSP 2.1 control mode allows a Windows application to control calls and terminate packetized voice on an associated IP phone. This mode is often referred to as “third-party call control”.

Figure 2 *Windows Telephony Architecture and Cisco Unified CME TAPI Components in Emulation Mode*



Cisco Unified CME TSP 2.1 emulation mode allows a Windows application to become an IP phone on your PC using an appropriate Windows driver. In emulation mode, the Cisco Unified CME TSP 2.1 application terminates the media. This mode is often referred to as “first-party call control.”

Call Control

You can configure Cisco Unified CME TSP 2.1 to provide either first- or third-party call control.

First-Party Call Control

In first-party call control, the application terminates the packetized audio stream. Cisco Unified CME TSP 2.1 uses Windows drivers to stream the audio packets to the selected audio devices when the call is connected. The result is call progress tones and the alerting tones are delivered to the specified audio devices.



Note

You must specify playback and recording audio devices using the Cisco Unified CME TSP 2.1 Setup Wizard. For more information see the *Cisco Unified CME Telephony Service Provider 2.1 Setup Guide* available at www.cisco.com.

In first-party configuration, the Windows application using Cisco Unified CME TSP 2.1 essentially becomes an IP softphone and looks like an IP phone to Cisco Unified CME.

Third-Party Call Control

In third-party call control, the audio stream is terminated on an IP phone on your desk and the TAPI application provides the remote control of calls to that IP phone. Call progress and alerting tones are not played for the calls.

Multiple TAPI Applications

In the Cisco Unified CME TSP 2.1 solution, the TAPI application and Cisco Unified CME TSP 2.1 get installed on the same machine. The TAPI application and Cisco Unified CME TSP 2.1 do not directly interface with each other. A layer written by Microsoft sits between the TAPI application and the Cisco Unified CME TSP 2.1. This layer, known as TAPISRV, allows the installation of multiple TSPs on the same machine, and it hides that fact from the 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 CME TSP 2.1 exposes 6 lines, and Microsoft H323 TSP exposes another line. The TAPI application has access to and control of both the Cisco Unified CME lines and the H323 line. The application communicates with TAPISRV, and TAPISRV takes care of communicating with the correct TSP.

Compatibility

Cisco Unified CME TSP 2.1 serves as a TAPI 2.2 service provider. When developing an application, be sure to use only functions that Cisco Unified CME TSP 2.1 supports. For example, transfer is supported, but fax detection is not. If an application requires a media or bearer mode that is not supported, it will not work as expected.

**Note**

Cisco Unified CME TSP 2.1 does not support the Call Center and Agent functions of the TAPI 2.2 specification.

Supported Device and Line Types

Cisco Unified CME TSP 2.1 supports the following device types:

- Cisco Unified IP Phone 7902
- Cisco Unified IP Phone 7905
- Cisco Unified IP Phone 7906G
- Cisco Unified IP Phone 7911G
- Cisco Unified IP Phone 7914 Expansion Module
- Cisco Unified IP Phone 7920
- Cisco Unified IP Phone 7940
- Cisco Unified IP Phone 7941G and 7941GE
- Cisco Unified IP Phone 7960
- Cisco Unified IP Phone 7961G and 7961GE
- Cisco Unified IP Phone 7970G
- Cisco Unified IP Phone 7971G
- Cisco IP Communicator 2.0.1.1

The TSP supports the following types of lines or DN configured on the ephone:

- Single Line DNs—Single line DNs allow a single instance of a call to be associated with the DN.
- Dual Line DNs—Dual line DNs allow two calls to be associated with the DN.
- Shared lines—A DN is shared when it is configured on more than one ephone. Incoming calls are presented to all the ephones. When the call is connected at one position, all other ephones are provided a “Line in Use” status. For shared lines that are “dual”, the second call is presented only to the ephone with the connected call.
- Monitored DNs—When a DN is monitored at an ephone, its busy/idle status is presented at the endpoint. The endpoint cannot use the Monitored DN to make or answer calls.

Startup with Windows

Cisco Unified CME TSP 2.1, as with other TSPs, launches at Windows startup. Even though Cisco Unified CME TSP 2.1 is running, it is normally not connected to Cisco Unified CME until a Windows application opens a session. Most TAPI applications open a TAPI session when the application is started and close it when the application terminates. However certain applications, such as Microsoft Outlook, open and close a TAPI session when a call is made. Cisco Unified CME TSP 2.1 provides a “Startup with Windows” mode in which it registers or “connects” to Cisco Unified CME at Windows startup and does not un-register until Windows shuts down. This “Startup with Windows” mode makes calling from such applications much faster.

Resets and Restarts

When Cisco Unified CME TSP 2.1 receives a Reset or Restart message from Cisco Unified CME, it immediately disconnects, removes all lines, and sends a `LINE_CLOSE` message to the Windows application indicating that the Cisco Unified CME session has been terminated. Both the Reset and Restart messages are treated in an identical manner.

**Note**

Certain other failure conditions, including loss of network connection and loss of a media audio device, can also cause the TSP to send a `LINE_CLOSE` message.

For a restart or reset, the TSP polls Cisco Unified CME to determine if it is ready to accept a connection. Upon receipt of confirmation, it sends a `LINE_REINIT` message to the application. This is an indication to the application that the TAPI session can be re-opened. If the “Startup with Windows” option is enabled, then the TSP automatically connects to the configured Cisco Unified CME endpoint. Otherwise, the TAPI session remains closed until the application re-opens the line.

Debug Tracing

Cisco Unified CME TSP 2.1 supports five levels of debug tracing. Debug traces are enabled from the Cisco Unified CME TSP 2.1 Setup Wizard. The trace levels are as follows:

- **Trace Level 0**—Tracing is turned off. The Trace Level should be set to zero in normal operation to avoid accumulating large trace files.
- **Trace Level 1**—Provides trace of TAPI function calls and messages.
- **Trace Level 2**—Provides TSP level of debug trace messages, including TAPI function calls and messages.
- **Trace Level 3**—Provides TSP level of debug trace messages and summary of Skinny (SCCP) messages.
- **Trace Level 4**—Provides TSP level of debug trace messages and Skinny message details.
- **Trace Level 5**—Provides Trace Level 4 and RTP transmission details.

**Note**

Levels 4 and 5 are very verbose and can quickly accumulate large amounts of log data. These modes should only be used at the direction of technical support.

Exception Notes

The following notes relate to some of the exceptions and idiosyncratic behavior you may encounter with Cisco Unified CME TSP 2.1.

Call Origin

The call origin information indicating an internal or external call is determined from the ring type message — Inside Ring or Outside Ring. When there are multiple calls, the TSP associates the ring-type message with the oldest ringing call. In some cases this association may not be correct. In cases where the call origin cannot be determined it is set to external by default.

Line Type

The Cisco Unified CME TSP 2.1 Setup Wizard automatically downloads the line configuration from Cisco Unified CME during the registration process. This configuration information only provides the line name and number. By default the TSP categorizes all the lines as dual-line DNs. For proper operation of the TSP the line types need to be manually set up to correctly reflect the configuration of the DN on the ephone.

Shared Dual Lines

The Cisco Unified CME call-handling model allows only the active ephone of shared dual line DN to receive additional calls. That is, if an ephone is active on a call on a shared DN, other ephones with the same DN will see the DN in use. If a second call is made to the dual-line DN, that call is only presented to the active ephone. The other ephone does not receive any indication of the second call.

Outgoing Calls on Shared Lines

When an outgoing call is made from a shared DN, the “shares” do not receive dialed number information. Therefore the TSP can only provide the call state “Outgoing call” information, but not any name or number.

MAC Address/Device ID

Cisco Unified CME requires a unique ID in the MAC Address format for each ephone. It does not require this to be the actual MAC Address of the PC or endpoint. In phones, this Device ID is normally the MAC Address of the phone. However for the TSP, this can be any number.

**Note**

If a device ID is entered for the TSP, during the registration process Cisco Unified CME matches this device ID to its configuration files. If there is no ephone configured with this device ID, an ephone configuration is automatically created. There are no lines configured for this ephone, however. If incorrect MAC address or device IDs are entered during the TSP configuration process, you must manually remove them from the Cisco Unified CME configuration.



Cisco Unified CME TAPI Line Device

Revised: January 12, 2007

The Cisco Unified CME TAPI implementation comprises a set of classes that expose the call handling functionality of Cisco Unified CME IP phone to Windows applications. This API allows developers to create customized IP telephony applications for Cisco Unified CME without specific knowledge of the communication protocols between the Cisco Unified CME and the service provider. For example, a developer could create a TAPI application that provides a screen-based call management adjunct to an IP phone.

This chapter outlines the TAPI 2.2 functions, events, and messages that Cisco Unified CME TSP 2.1 supports. The Cisco Unified CME TAPI implementation contains functions in the following areas:

- [Cisco Unified CME TAPI Line Functions, page 11](#)
- [Cisco Unified CME TAPI Line Messages, page 44](#)
- [Cisco Unified CME TAPI Line Structures, page 55](#)

Cisco Unified CME TAPI Line Functions

Cisco Unified CME TSP 2.1 supports only one IP phone device. It supports both “first-party” call control that allows the application to terminate the media and “third-party” call control that allows an application to control the calls made or received on the associated IP phone.

This chapter documents the function calls handled by the TSP. For information about the TAPI calls handled by TAPISRV, please refer to the Microsoft documents.

lineAddtoConference

Description

The lineAddtoConference function takes the consultation 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 consultation 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 consultation call must be in the connected state.

Return Values

Possible return values are:

```
LINEERR_INVALCONFCALLHANDLE, LINEERR_OPERATIONUNAVAIL, LINEERR_INVALCALLHANDLE,
LINEERR_OPERATIONFAILED, LINEERR_INVALCALLSTATE,
```

Further Details

If LINEERR_INVALCALLHANDLE is returned, the specified call handle for the added call is invalid. The call handle of the added party remains valid after adding the call to a conference. Its state typically changes to conferenced while the state of the conference call typically becomes connected.

lineAnswer

Description

The lineAnswer function answers the specified offering call.



Note

Cisco Unified CME places the previous call on the device in the connected call state on hold before answering the new call. If the previous call is not in the connected state (such as when ringing or dialing), then that call can be dropped.

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.¹

`dwSize`

The size in bytes of the user-user information in `lpsUserUserInfo`. If `lpsUserUserInfo` is NULL, no user-user information is sent to the calling party, and `dwSize` is ignored.¹

Return Values

`LINEERR_INVALIDCALLHANDLE`, `LINEERR_OPERATIONFAILED`, `LINEERR_INVALIDCALLSTATE`.

Further Details

When a new call arrives, applications with an interest in the call are sent a `LINE_CALLSTATE` message to provide the new call handle and to inform the application about the call's state and the privileges to the new call (such as monitor or owner). The application with owner privilege for the call can answer this call using `lineAnswer`. After the call has been successfully answered, the call typically transitions to the connected state.

If a call comes in (is offered) at the time another call is already active, invoking `lineAnswer` connects to the new call. The effect this has on the existing active call depends on the line's device capabilities. The first call can be unaffected, it can automatically be dropped, or it can automatically be placed on hold. The appropriate `LINE_CALLSTATE` messages report state transitions to the application about both calls.

lineBlindTransfer

Description

The `lineBlindTransfer` function performs a blind or single-step transfer of the specified call to the specified destination address.



Note

When the CME is configured for Consultation-Transfer, the `lineBlindTransfer` function is implemented as a single-step transfer in that the TSP automatically sends Transfer-Complete.

1. Cisco Unified CME TSP 2.1 does not support user-user information. This should be set to NULL.

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.

Return Values

`LINEERR_INVALIDCALLHANDLE`, `LINEERR_INVALIDCALLSTATE`, `LINEERR_OPERATIONFAILED`.

Further Details

Blind transfer differs from a consultation transfer in that no consultation call is made visible to the application. After the blind transfer successfully completes, the specified call is typically cleared from the application's line, and it transitions to the idle state.

The application's call handle remains valid after the transfer has completed. The application must deallocate its handle using `lineDeallocateCall` when it is no longer interested in the transferred call. If the consultation call fails, and does not ring back, then transfer does not complete and the application is responsible for clearing all related call handles.

lineCallbackFunc

Description

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 provided by dwMsg 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 passed to this function, see the [“Cisco Unified CME TAPI Line Messages” section on page 44](#).

All callbacks occur in the application’s context. The callback function must reside in a DLL or application module.

lineClose

Description

The lineClose function closes the specified open line device.

Function Details

```
LONG lineClose(
    HLINE hLine
);
```

Parameters

hLine

A handle to the open line device to be closed. After the line has been successfully closed, this handle is no longer valid.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs. A possible return values is:

```
LINEERR_INVALIDLINEHANDLE.
```

Further Details

If an application calls lineClose while it still has active calls on the opened line, the application's ownership of these calls is revoked. If the application was the sole owner of these calls, the calls are dropped as well. It is good programming practice for an application to dispose of the calls it owns on an opened line by explicitly relinquishing ownership and/or by dropping these calls prior to closing the line.

If the line was closed successfully, a LINE_LINEDEVSTATE message is sent to all applications that are monitoring the line status of open/close changes. Outstanding asynchronous replies are suppressed.

lineCompleteCall

Description

The lineCompleteCall function specifies how a call that could not be connected normally should be completed instead. The network or switch may not be able to complete a call because network resources are busy or the remote station is busy or doesn't answer. The application can request that the call be completed in one of a number of ways.

Cisco Unified CME TSP 2.1 supports only the Callback completion mode.

Function Details

```
LONG WINAPI lineCompleteCall(
    HCALL hCall,
    LPDWORD lpwdCompletionID,
    DWORD dwCompletionMode,
    DWORD dwMessageID
);
```

Parameters

`hCall`

A handle to the call whose completion is requested. The application must be an owner of the call. The call state of `hCall` must be busy, ringback.

`lpdwCompletionID`

A pointer to a DWORD-sized memory location. The completion identifier is used to identify individual completion requests in progress. A completion identifier becomes invalid and can be reused after the request completes or after an outstanding request is canceled.

`dwCompletionMode`

The way in which the call is to be completed. This parameter uses one and only one of the `LINECALLCOMPLMODE_` constants.



Note

Only `LINECALLCOMPLMODE_CALLBACK` is supported by Cisco Unified CME TSP 2.1.

`dwMessageID`

The message that is to be sent when completing the call using `LINECALLCOMPLMODE_MESSAGE`. This identifier selects the message from a small number of predefined messages.

Return Values

`LINEERR_INVALIDCALLCOMPLMODE`, `LINEERR_INVALIDCALLHANDLE`, `LINEERR_OPERATIONFAILE`.

Further Details

This function is considered complete when the request has been accepted by the network or switch; not when the request is fully completed in the way specified. After this function completes, the call typically transitions to idle.

lineCompleteTransfer

Description

The `lineCompleteTransfer` function completes the transfer of the specified call to the party that is connected in the consultation call.



Note

Cisco Unified CME TSP 2.1 only supports the transfer operation—not conference.

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 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, this parameter is ignored by TAPI.

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 is resolved by establishing a three-way conference between the application, the party connected to the initial call, and the party connected to the consultation call. Selecting this option creates a conference call.

Return Values

```
LINEERR_INVALIDCALLHANDLE, LINEERR_INVALIDCALLSTATE, LINEERR_OPERATIONUNAVAIL,
LINEERR_INVALIDCONSULTCALLHANDLE, LINEERR_OPERATIONFAILED, LINEERR_INVALIDTRANSFERMODE.
```

Further Details

The LINE_REPLY message sent in response to a call to the lineCompleteTransfer function is based on the status of the call specified by the hCall parameter.

This operation completes the transfer of the original call, hCall, to the party currently connected by hConsultCall. The consultation call is typically dialed on the consultation call allocated as part of lineSetupTransfer.

The transfer request can only be resolved as a transfer; completion as three-way conference call is not supported. When resolved as a transfer, the parties connected by hCall and hConsultCall are connected to each other, and both hCall and hConsultCall are typically cleared from the application's line and transition to the idle state. The application's call handle remains valid after the transfer has completed. The application must deallocate its handle with lineDeallocateCall when it is no longer interested in the transferred call.

lineConfigProvider

Description

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 be `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 error number if an error occurs. Possible return values follow:

```
LINEERR_INVALIDPARAM,  
LINEERR_OPERATIONFAILED.
```

lineDeallocateCall

Description

The `lineDeallocateCall` function deallocates the specified call handle.

Function Details

```
LONG lineDeallocateCall(  
    HCALL hCall  
);
```

Parameters

`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 no longer valid after it has been deallocated.

Return Values

`LINEERR_INVALIDCALLHANDLE, LINEERR_INVALIDCALLSTATE.`

lineDevSpecificFeature

Description

The `lineDevSpecificFeature` function enables service providers to provide access to features not offered by other TAPI functions. The meaning of these extensions are device specific, and taking advantage of these extensions requires the application to be fully aware of them.

When used with Cisco Unified CME TSP 2.1, `lineDevSpecific` can be used to:

- Send keypad digits
- Send line button selection
- Play audio file
- Stop playing audio file

Function Details

```
LONG lineDevSpecific(
    HLINE hLine,
    DWORD dwAddressID,
    HCALL hCall,
    LPVOID lpParams,
    DWORD dwSize
);
```

Parameters

`hLine`

A handle to the line device.

`dwFeature`

The feature to invoke on the line device. This parameter uses the `PHONEBUTTONFUNCTION_` constants.

The following PHONEBUTTONFUNCTIONS are defined:

```
PHONEBUTTONFUNCTION_PRESSBUTTON,  
PHONEBUTTONFUNCTION_PLAYFILE,  
PHONEBUTTONFUNCTION_STOPFILE,  
PHONEBUTTONFUNCTION_MONITOR.
```

```
lpParams
```

A pointer to a memory area used to hold a feature-dependent parameter block. The format of this parameter block is device specific and its contents are passed through by TAPI to or from the service provider.

```
dwSize
```

The size of the buffer in bytes.

Return Values

```
LINEERR_INVALIDFEATURE,  
LINEERR_INVALIDLINEHANDLE,  
LINEERR_OPERATIONFAILED
```

Additional return values are device specific.

lineDial

Description

The lineDial function dials the specified number on the specified call.

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. The application must be 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.

Return Values

Possible return values are:

```
LINEERR_INVALIDCALLHANDLE, LINEERR_RESOURCEUNAVAIL, LINEERR_INVALIDCALLSTATE.
```

Further Details

The `lineDial` function is used for dialing on an existing call appearance. For example, after a call has been set up for transfer or conference, a consultation call is automatically allocated, and the `lineDial` function would be used to perform the dialing of this consultation call. The `lineDial` function can be invoked multiple times in the course of multistage dialing, if the line's device capabilities allow it.

Dialing is considered complete after the address has been passed to the service provider; not after the call is finally connected. The service provider sends `LINE_CALLSTATE` messages to the application to inform it about the progress of the call. To abort a call attempt while a call is being established, the invoking application should use `lineDrop`.

lineDrop

Description

The `lineDrop` function drops or disconnects the specified call.¹

Function Details

```
LONG lineDrop(
    HCALL hCall,
    LPCSTR lpsUserUserInfo,
    DWORD dwSize
);
```

Parameters

`hCall`

A handle to the call to be dropped. The application must be an owner of the call. The call state of `hCall` can be any state except idle.

`lpsUserUserInfo`

A pointer to a string that contains user-user information to be sent to the remote party as part of the call disconnect. This pointer can be left `NULL` if no user-user information is to be sent.¹

`dwSize`

The size in bytes of the user-user information in `lpsUserUserInfo`. If `lpsUserUserInfo` is `NULL`, no user-user information is sent to the calling party, and `dwSize` is ignored.¹

1. Cisco Unified CME TSP 2.1 does not support user-user information. This should be set to `NULL`.

Return Values

Possible return values are:

```
LINEERR_INVALIDCALLHANDLE, LINEERR_INVALIDCALLSTATE.
```

Further Details

When invoking `lineDrop`, related calls can sometimes be affected as well. For example, dropping a conference call can drop all individual participating calls. `LINE_CALLSTATE` messages are sent to the application for all calls whose call state is affected. A dropped call typically transitions to the idle state.

lineForward

Description

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 currently is 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

The `lineForward` implementation of Cisco Unified CME TSP 2.1 allows setting 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 are to be forwarded. If `TRUE`, all addresses on the line get forwarded, and `dwAddressID` is ignored; if `FALSE`, only the address that is specified as `dwAddressID` is forwarded.

`dwAddressID`

The address of the specified line whose incoming calls are to be forwarded. This parameter is ignored if `bAllAddresses` is `TRUE`.



Note

If `bAllAddresses` is `FALSE`, `dwAddressID` must be 0.

`lpForwardList`

A pointer to a variably sized data structure that describes the specific forwarding instructions of type `LINEFORWARDLIST`.

`lpForwardList`

To cancel forwarding, ensure this 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 is set to the nearest value in the allowable range.



Note

This parameter is not used because this version of Cisco Unified CME TSP 2.1 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 that is being forwarded to, 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 ignored because Cisco Unified CME TSP 2.1 does not use a consultation call to set up `lineForward`.

`lpCallParams`

A pointer to a structure of type `LINECALLPARAMS`. This pointer is 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` because Cisco Unified CME TSP 2.1 does not create a consultation call.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs.

Possible return values follow:

```
LINEERR_INVALLINEHANDLE,
LINEERR_INVALADDRESSID,
LINEERR_INVALADDRESS, LINEERR_OPERATIONFAILED,
```

```
LINEERR_INVALIDPARAM, LINEERR_UNINITIALIZED.
```

**Note**

For `lpForwardList[0].dwForwardMode` other than `UNCOND`, `lineForward` returns `LINEERR_OPERATIONUNAVAIL`. For `lpForwardList.dwNumEntries` more than 1, `lineForward` returns `LINEERR_INVALIDPARAM`.

lineGetAddressCaps

Description

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 is 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 telephony 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 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 is 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.

Further Details

The following address capabilities are returned by this call:

Address Sharing

```
LINEADDRESSSHARING_PRIVATE |
LINEADDRESSSHARING_BRIDGEEXCL;
```

Address States

```
LINEADDRESSSTATE_INUSEZERO |
LINEADDRESSSTATE_INUSEONE |
LINEADDRESSSTATE_NUMCALLS;
```

Call Info States

```
LINECALLINFOSTATE_APPSPECIFIC |
LINECALLINFOSTATE_ORIGIN |
LINECALLINFOSTATE_REASON;
```

Call States

```
LINECALLSTATE_IDLE |
LINECALLSTATE_OFFERING |
LINECALLSTATE_ACCEPTED |
LINECALLSTATE_DIALTONE |
LINECALLSTATE_DIALING |
LINECALLSTATE_RINGBACK |
LINECALLSTATE_BUSY |
LINECALLSTATE_CONNECTED |
LINECALLSTATE_PROCEEDING |
LINECALLSTATE_ONHOLD |
LINECALLSTATE_CONFERENCED |
LINECALLSTATE_ONHOLDPENDCONF |
LINECALLSTATE_ONHOLDPENDTRANSFER |
LINECALLSTATE_DISCONNECTED;
```

Line Address Features

```
LINEADDRFEATURE_SETMEDIACONTROL |
LINEADDRFEATURE_SETTERMINAL |
LINEADDRFEATURE_FORWARD |
LINEADDRFEATURE_PICKUP |
LINEADDRFEATURE_PICKUPDIRECT |
LINEADDRFEATURE_PICKUPGROUP |
LINEADDRFEATURE_UNPARK |
LINEADDRFEATURE_SETUPCONF;
```

Call Features

```
LINECALLFEATURE_ACCEPT |
LINECALLFEATURE_ADDTOCONF |
```

```

LINECALLFEATURE_ANSWER |
LINECALLFEATURE_COMPLETETRANSF |
LINECALLFEATURE_DIAL |
LINECALLFEATURE_DROP |
LINECALLFEATURE_HOLD |
LINECALLFEATURE_PARK |
LINECALLFEATURE_PREPAREADDCONF |
LINECALLFEATURE_REMOVEFROMCONF |
LINECALLFEATURE_SETUPCONF |
LINECALLFEATURE_SETUPTRANSFER |
LINECALLFEATURE_UNHOLD |
LINECALLFEATURE_COMPLETECALL |
LINECALLFEATURE_BLINDTRANSFER;

```

Call Features2

```
LINECALLFEATURE2_COMPLCALLBACK ;
```

Remove From Conf State

```
LINECALLSTATE_IDLE;
```

Transfer Modes

```
LINETRANSFERMODE_TRANSFER |
```

lineGetAddressStatus

Description

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 is 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.

Return Values

```
LINEERR_RESOURCEUNAVAIL,  
LINEERR_INVALIDADDRESSID;
```

lineGetCallInfo

Description

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.

Return Values

```
LINEERR_INVALIDAPPHANDLE;
```

lineGetCallStatus

Description

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.

Return Values

`LINEERR_INVALIDAPPHANDLE;`

lineGetDevCaps

Description

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 telephony 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 left 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 is 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

Description

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 the device class API uses for naming devices. Prior to calling lineGetID, the application should set the dwTotalSize member of this structure to indicate the amount of memory that is available to TAPI for returning information.

```
lpzDeviceClass
```

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.

Return Values

```
LINEERR_OPERATIONFAILED,  
LINEERR_INVALIDAPPHANDLE,  
LINEERR_INVALIDPOINTER, LINEERR_STRUCTURETOOSMALL, LINEERR_NODEVICE;
```

lineHold

Description

The lineHold function places the specified call on hold.

Function Details

```
LONG lineHold(  
HCALL hCall  
);
```

Parameters

```
hCall
```

A handle to the call that is to be placed on hold. Ensure the application is an owner of the call and the call state of hCall is connected.

Return Values

```
LINEERR_INVALIDAPPHANDLE,  
LINEERR_INVALIDCALLSTATE;
```

lineInitializeEx

Description

The `lineInitializeEx` function initializes TAPI 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 hand-off 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 is 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 is 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 is filled with the number of line devices that are available to the application.

`lpdwAPIVersion`

A pointer to a DWORD-sized location. Before calling this function, the application must initialize this DWORD 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; the value must be 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, `0x00020000`, supports thereby allowing the application to detect and adapt to having been installed on a system with an older version of TAPI.

`lpLineInitializeExParams`

A pointer to a structure of type `LINEINITIALIZEEXPARAMS` which 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

Description

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
);
```

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 invoking other telephony operations on the call. The application initially acts as the sole owner of this call. This handle registers as void if the function returns an error (synchronously or asynchronously by the reply message).

`lpzDestAddress`

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. 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 is non-zero, used to automatically disconnect a call if it is not answered after the specified time.

Return Values

```
LINEERR_INVALIDBEARERMODE,
LINEERR_INVALIDAPPHANDLE,
LINEERR_INVALIDPOINTER, LINEERR_INVALIDMEDIAMODE,
LINEERR_INVALIDCALLPARAMS,
LINEERR_CALLUNAVAIL,
LINEERR_NODEVICE;
```

lineNegotiateAPIVersion

Description

The `lineNegotiateAPIVersion` function allows an application to negotiate an API version to use. Cisco Unified CME TSP 2.1 supports TAPI 2.2.

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 is 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.

Return Values

`LINEERR_INCOMPATIBLEAPIVERSION`

lineOpen

Description

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 either can be a valid device identifier or the value `LINEMAPPER`.

**Note**

Cisco Unified CME TSP 2.1 does not support LINEMAPPER. LphLine is a pointer to an HLINE handle that is then loaded with the handle representing 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 2.1 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 it is not an owner of a call, it is a monitor.

`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 hand-off 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 `LINEMEDIAMODE_INTERACTIVEVOICE` constant where 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.

`lpCallParams`

The `dwNoAnswerTimeout` attribute of the `lpCallParams` field is checked, and if it is non-zero, used to automatically disconnect a call that is not answered after the specified time.

Return Values

`LINEERR_RESOURCEUNAVAIL,`
`LINEERR_OPERATIONUNAVAIL,`

```
LINEERR_BADEVICEID;
```

linePark

Description

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 to be parked. This parameter can have only a single flag set and uses one of the LINEPARKMODE_ constants.

```
LINEPARKMODE_DIRECTED
LINEPARKMODE_NONDIRECTED
```



Note

Cisco Unified CME TSP 2.1 transfers a call to the number supplied in DirAddress when the mode is set to LINEPARK_DIRECTED. The Cisco Unified CME park function is called when the park mode is set to LINEPARKMODE_NONDIRECTED. All address information is ignored for park.

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 is ignored for nondirected park.



Note

This parameter is ignored.

lpNonDirAddress

Pointer to a structure of type VARSTRING. For nondirected park, the address where the call is parked is returned in this structure. This parameter is ignored for directed park. Within the VARSTRING structure, dwStringFormat must be 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.

Return Values

`LINEERR_INVALIDCALLHANDLE,`

linePickup

Description

The `linePickup` function picks up a call alerting at the specified destination address and returns a call handle for the picked-up call. If invoked with NULL for the `lpszDestAddress` parameter, a group pickup is performed. If required by the device, `lpszGroupID` specifies the group identifier to which the alerting station belongs.

Function Details

```
LONG WINAPI linePickup(
    HLINE hLine,
    DWORD dwAddressID,
    LPHCALL lphCall,
    LPCSTR lpszDestAddress,
    LPCSTR lpszGroupID
);
```

Parameters

`hLine`

Handle to the open line device on which a call is to be picked up.

`dwAddressID`

Address on `hLine` at which the pickup is to be originated. An address identifier is permanently associated with an address; the identifier remains constant across operating system upgrades.

`lphCall`

Pointer to a memory location where the handle to the picked up call is returned. The application is the initial sole owner of the call.

`lpszDestAddress`

Pointer to a null-terminated character buffer that contains the address whose call is to be picked up. The address is in standard dialable address format.

`lpszGroupID`

Pointer to a null-terminated character buffer containing the group identifier to which the alerting station belongs. This parameter is required on some switches to pick up calls outside of the current pickup group.

The `lpszGroupID` parameter can be specified by itself with a NULL pointer for `lpszDestAddress`. Alternatively, `lpszGroupID` can be specified in addition to `lpszDestAddress`, if required by the device.

Return Values

```
LINEERR_INVALIDADDRESS, LINEERR_NOMEM, LINEERR_INVALIDADDRESSID, LINEERR_OPERATIONUNAVAIL,
LINEERR_INVALIDGROUPID, LINEERR_OPERATIONFAILED, LINEERR_INVALIDLINEHANDLE,
LINEERR_RESOURCEUNAVAIL, LINEERR_INVALIDPOINTER, LINEERR_UNINITIALIZED.
```

Further Details

When a call has been picked up successfully, the application is notified by the `LINE_CALLSTATE` message about call state changes. The `LINECALLINFO` structure supplies information about the call that was picked up. It lists the reason for the call as pickup. This structure is available using `lineGetCallInfo`.

lineRemoveProvider

Description

The `lineRemoveProvider` function removes an existing telephony service provider from the telephony 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

Possible return values follow:

```
LINEERR_INIFILECORRUPT, LINEERR_NOMEM,
LINEERR_INVALIDPARAM, LINEERR_OPERATIONFAILED.
```

lineSetupConference

Description

The `lineSetupConference` function initiates a conference given an existing two-party call that the `hCall` parameter specifies. A conference call and consultation call are established and the handles return to the application. Use a consultation call to dial the third party and the conference call replaces the initial two-party call.

Function Details

```
LONG lineSetupConference (
    HCALL hCall,
    HLINE hLine,
    LPHCALL lphConfCall,
    LPHCALL lphConsultCall,
    DWORD dwNumParties,
    LPLINECALLPARAMS const lpCallParams
);
```

Parameters

`hCall`

The handle of the call to be transferred. The application must be an owner of the call. The call state of `hCall` must be 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 is automatically allocated to enable `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 to be transferred. It may also be necessary that the consultation call be set up 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.

Return Values

```
LINEERR_CALLUNAVAIL, LINEERR_INVALIDPOINTER, LINEERR_INVALIDCALLHANDLE,
LINEERR_OPERATIONUNAVAIL, LINEERR_INVALIDCALLSTATE, LINEERR_OPERATIONFAILED,
```

lineSetupTransfer

Description

The `lineSetupTransfer` function initiates a transfer of the call that the `hCall` parameter specifies. It establishes a consultation call, `lphConsultCall`, to the party who can become the transfer destination. 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. The application must be an owner of the call. The call state of `hCall` must be 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 is automatically allocated to enable `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 to be transferred. It may also be necessary that the consultation call be set up 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.

Return Values

```
LINEERR_CALLUNAVAIL,  
LINEERR_INVALIDCALLHANDLE, LINEERR_OPERATIONUNAVAIL, LINEERR_INVALIDCALLSTATE,  
LINEERR_INVALIDLINEHANDLE,
```

lineUnhold

Description

The lineUnhold function retrieves a 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.

Return Values

```
LINEERR_INVALIDCALLHANDLE, LINEERR_INVALIDCALLSTATE,
LINEERR_OPERATIONFAILED,
```

lineUnpark

Description

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.

Return Values

`LINEERR_INVALLINEHANDLE,`

Cisco Unified CME TAPI Line Messages

This section describes the line messages that Cisco Unified CME TSP 2.1 supports. These messages notify the application of asynchronous events such as the a new call arriving at Cisco Unified CME. The messages are sent to the application using the method that the application specifies in `lineInitializeEx`.



Note

TAPI Line Messages

LINE_ADDRESSSTATE
 LINE_APPNEWCALL
 LINE_CALLINFO
 LINE_CALLSTATE
 LINE_CLOSE
 LINE_CREATE
 LINE_DEVSPECIFICFEATURE
 LINE_LINEDEVSTATE
 LINE_REMOVE
 LINE_REPLY
 LINE_REQUEST

LINE_ADDRESSSTATE

Description

The `LINE_ADDRESSSTATE` message is 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 that 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` — Changed address-status items other than those listed below. The application should check the current address status to determine which items changed.
- `LINEADDRESSSTATE_DEVSPECIFIC` — Device-specific item of the changed address status.
- `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 address's current forwarding status.
- `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 re-initialize their connection to TAPI to obtain the updated information.

`dwParam3`

Not used.

LINE_APPNEWCALL

Description

The `LINE_APPNEWCALL` message informs an application when a new call handle was 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_CALLINFO

Description

The TAPI `LINE_CALLINFO` message is 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`

Not used.

`dwParam3`

Not used.

LINE_CALLSTATE

Description

The LINE_CALLSTATE message is sent when the status of the specified call changes. Typically, several such messages are received during the lifetime of a call. Applications get notified of new incoming calls with this message; the new call is 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 belonging to this call is opened.

dwParam1

The new call state.

Cisco Unified TSP only supports the following LINECALLSTATE_ values:

- LINECALLSTATE_IDLE — The call is idle; no call actually exists.
- LINECALLSTATE_OFFERING — The call is being offered to the station, signaling the arrival of a new call. In some environments, a call in the offering state does not automatically alert the user. The switch instructing the line to ring does alerts; it does not affect any call states.
- LINECALLSTATE_ACCEPTED — The call was offering and has been accepted. This indicates to other (monitoring) applications that the current owner application has claimed responsibility for answering the call. In ISDN, this also indicates that alerting to both parties has started.
- LINECALLSTATE_CONFERENCED — The call is a member of a conference call and is logically in the connected state.
- LINECALLSTATE_DIALTONE — The call is receiving 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 being sent to the switch over the call. The lineGenerateDigits does not place the line into the dialing state.
- LINECALLSTATE_RINGBACK — The call is receiving ringback from the called address. Ringback indicates that the other station has been reached and is being alerted.
- LINECALLSTATE_ONHOLDPENDCONF — The call is currently on hold while it is being added to a conference.

- `LINECALLSTATE_CONNECTED` — The call has been established and the connection is made. Information can flow over the call between the originating address and the destination address.
- `LINECALLSTATE_PROCEEDING` — Dialing completed, and the call is proceeding through the switch or telephone network.
- `LINECALLSTATE_ONHOLD` — The call is on hold by the switch.
- `LINECALLSTATE_ONHOLDPENDTRANSFER` — The call is currently on hold awaiting 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 be due to limitations of the call-progress detection implementation.

**Note**

If application negotiates extension version 0x00050001 or greater can receive device-specific `CLDSMT_CALL_PROGRESSING_STATE = 0x01000000` with `LINECALLSTATE_UNKNOWN`. This is a device-specific TAPI call state supported by Cisco Unified CME.

`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` — The call is connected at the current station (the current station acts as a participant in the call).
- `LINECONNECTEDMODE_INACTIVE` — The call is active at one or more other stations, but the current station is not a participant in the call. When a call is disconnected with cause code = `DISCONNECTMODE_TEMPFAILURE` and the `lineState = LINEDEVSTATE_INSERTSERVICE`, applications must take care of dropping the call. If the application is terminating media for a device, then it is also the responsibility of the application to stop the RTP streams for the same call. The 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 failed call on the 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`, then the cause code will be reported as the standard Q931 cause codes in `dwParam2`.

If `dwParam1` is `LINECALLSTATE_DIALTONE`, `dwParam2` contains the details about the dial tone mode.

This parameter uses the `LINEDIALTONEMODE_UNAVAIL` constant, where the dial tone mode is unavailable and cannot become known. If `dwParam1` is `LINECALLSTATE_OFFERING`, `dwParam2` contains details about the connected mode.

This parameter uses the `LINEOFFERINGMODE_ACTIVE` constant, where 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 is the situation on a non-bridged address).

**Note**

Cisco Unified CME TSP 2.1 does not send `LINEDEVSTATE_RINGING` messages until the call is accepted and moves to the `LINECALLSTATE_ACCEPTED` state. IP phones auto-accept calls.

Computer telephony integration (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` is `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, the call terminated normally.
- `LINEDISCONNECTMODE_UNKNOWN` — The reason for the disconnect request is 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` — The network is congested.
- `LINEDISCONNECTMODE_UNAVAIL` — The reason for the disconnect is unavailable and cannot be determined later.
- `LINEDISCONNECTMODE_FACCMC` — The call has been disconnected by the Forced Authorization Code (FAC) and Client Matter Code (CMC) feature.

**Note**

`LINEDISCONNECTMODE_FACCMC` is only returned if the extension version negotiated on the line is 0x00050000 (5.0) or higher. If the negotiated extension version is not at least 0x00050000, then the TSP will set the disconnect mode to `LINEDISCONNECTMODE_UNAVAIL`.

`dwParam3`

If zero, this parameter indicates that there has not been a change in the privilege 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

Description

The `LINE_CLOSE` message is sent when the specified line device has been forcibly closed. The line device handle or any call handles for calls on the line are no longer valid after this message has been 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 belonging to this call is opened.

`dwParam1`

Not used.

`dwParam2`

Not used.

`dwParam3`

Not used.

LINE_CREATE

Description

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`

Not used.

dwCallbackInstance

Not used.

dwParam1

The dwDeviceID of the newly created device.

dwParam2

Not used.

dwParam3

Not used.

LINE_DEVSPECIFIC

Description

The LINE_DEVSPECIFIC message notifies the application about device-specific events that are occurring on a line, address, or call. The meaning of the message and the 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

A handle to either a line device or call. This is device specific.

dwCallbackInstance

The callback instance that is supplied when the line is opened.

dwParam1

Device-specific

dwParam2

Device-specific

dwParam3

Device-specific

LINE_DEVSTATE

Description

The TAPI LINE_LINEDEVSTATE message is 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.

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 the message was issued by TAPI 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 is a “real” REINIT message that requires the application to call lineShutdown at its earliest convenience.

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_REMOVE

Description

The `LINE_REMOVE` message informs an application of the removal (deletion from the system) of a line device. Generally, this parameter does not get used for temporary removals, such as extraction of PCMCIA devices, but only for permanent removals in which the device would no longer be reported by the service provider, if TAPI were re-initialized.

**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

Description

The `LINE_REPLY` message reports the results of function calls that completed asynchronously.

Function Details

```

LINE_REPLY
dwDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) idRequest;
dwParam2 = (DWORD) Status;
dwParam3 = (DWORD) 0;

```

Parameters

hDevice

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 does not get 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.

Cisco Unified CME TAPI Line Structures

This section describes the main line structures that are impacted by Cisco Unified CME TSP 2.1. These structures are used to exchange parameters between the application and TAPI and between TAPI and Cisco Unified CME TSP 2.1. In response to the line messages from the TSP, the TAPI layer makes functions calls with these structures to obtain the message-related detailed information.

LINEADDRESSCAPS

Description

The LINEADDRESSCAPS structure describes the capabilities of a specified address. The lineGetAddressCaps function and the TSPI_lineGetAddressCaps function return the LINEADDRESSCAPS structure.

Function Details

```
typedef struct lineaddresscaps_tag {
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    DWORD dwLineDeviceID;
    DWORD dwAddressSize;
    DWORD dwAddressOffset;
    DWORD dwDevSpecificSize;
    DWORD dwDevSpecificOffset;
    DWORD dwAddressSharing;
    DWORD dwAddressStates;
    DWORD dwCallInfoStates;
    DWORD dwCallerIDFlags;
    DWORD dwCalledIDFlags;
    DWORD dwConnectedIDFlags;
    DWORD dwRedirectionIDFlags;
    DWORD dwRedirectingIDFlags;
    DWORD dwCallStates;
    DWORD dwDialToneModes;
    DWORD dwBusyModes;
    DWORD dwSpecialInfo;
    DWORD dwDisconnectModes;
    DWORD dwMaxNumActiveCalls;
    DWORD dwMaxNumOnHoldCalls;
    DWORD dwMaxNumOnHoldPendingCalls;
    DWORD dwMaxNumConference;
    DWORD dwMaxNumTransConf;

    DWORD dwAddrCapFlags;
    DWORD dwCallFeatures;
    DWORD dwRemoveFromConfCaps;
    DWORD dwRemoveFromConfState;
    DWORD dwTransferModes;
    DWORD dwParkModes;
    DWORD dwForwardModes;
    DWORD dwMaxForwardEntries;
    DWORD dwMaxSpecificEntries;
    DWORD dwMinFwdNumRings;
    DWORD dwMaxFwdNumRings;
}
```

```

    DWORD   dwMaxCallCompletions;
    DWORD   dwCallCompletionConds;
    DWORD   dwCallCompletionModes;
    DWORD   dwNumCompletionMessages;
    DWORD   dwCompletionMsgTextEntrySize;
    DWORD   dwCompletionMsgTextSize;
    DWORD   dwCompletionMsgTextOffset;
    DWORD   dwAddressFeatures;
    DWORD   dwPredictiveAutoTransferStates;
    DWORD   dwNumCallTreatments;
    DWORD   dwCallTreatmentListSize;
    DWORD   dwCallTreatmentListOffset;
    DWORD   dwDeviceClassesSize;
    DWORD   dwDeviceClassesOffset;
    DWORD   dwMaxCallDataSize;
    DWORD   dwCallFeatures2;
    DWORD   dwMaxNoAnswerTimeout;
    DWORD   dwConnectedModes;
    DWORD   dwOfferingModes;
    DWORD   dwAvailableMediaModes;
} LINEADDRESSCAPS, FAR *LPLINEADDRESSCAPS;

```

Parameters

`dwTotalSize`

The total size, in bytes, 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.

`dwLineDeviceID`

The device identifier of the line device with which this address is associated.

`dwAddressSize`

`dwAddressOffset`

The size, in bytes, of the variably sized address field and the offset, in bytes, from the beginning of this data structure.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, of the variably sized device-specific field and the offset, in bytes, from the beginning of this data structure.

`dwAddressSharing`

The sharing mode of the address. This member can be one of the `LINEADDRESSSHARING_` constants.

`dwAddressStates`

Contains the address states changes for which the application may get notified in the `LINE_ADDRESSSTATE` message. This member uses one or more of the `LINEADDRESSSTATE_` constants.

`dwCallInfoStates`

Describes the call information elements that are meaningful for all calls on this address. An application may get notified about changes in some of these states in `LINE_CALLINFO` messages. This member uses one or more of the `LINECALLINFOSTATE_` constants.

`dwCallerIDFlags`

`dwCalledIDFlags`

`dwConnectedIDFlags`

`dwRedirectionIDFlags`

`dwRedirectingIDFlags`

Describes the various party identifier information types that can be provided for calls on this address. The caller is the originator of the session, “called” refers to the original destination, “redirection” is the new destination, and “redirecting” is the address which invoked redirection. These members uses one or more of the `LINECALLPARTYID_` constants.

`dwCallStates`

Describes the various call states that can be reported for calls on this address. This member uses one or more of the `LINECALLSTATE_` constants.

`dwDialToneModes`

Describes the various dial tone modes that can be reported for calls made on this address. This member is meaningful only if the dialtone call state can be reported. This member uses one or more of the `LINEDIALTONEMODE_` constants.

`dwBusyModes`

Describes the various busy modes that can be reported for calls made on this address. This member is meaningful only if the busy call state can be reported. This member uses one or more of the `LINEBUSYMODE_` constants.

`dwSpecialInfo`

Describes the various special information types that can be reported for calls made on this address. This member is meaningful only if the specialInfo call state can be reported. This member uses one or more of the `LINESPECIALINFO_` constants.

`dwDisconnectModes`

Describes the various disconnect modes that can be reported for calls made on this address. This member is meaningful only if the disconnected call state can be reported. This member uses one or more of the `LINEDISCONNECTMODE_` constants.

`dwMaxNumActiveCalls`

Contains the maximum number of active call appearances that the address can handle. This number does not include calls on hold or calls on hold pending transfer or conference.

`dwMaxNumOnHoldCalls`

Contains the maximum number of call appearances at the address that can be on hold.

`dwMaxNumOnHoldPendingCalls`

Contains the maximum number of call appearances at the address that can be on hold pending transfer or conference.

`dwMaxNumConference`

Contains the maximum number of parties that can join a single conference call on this address.

`dwMaxNumTransConf`

Specifies the number of parties (including “self”) that can be added in a conference call that is initiated as a generic consultation call using `lineSetupTransfer`.

`dwAddrCapFlags`

Contains a series of packed bit flags that describe a variety of address capabilities. This member uses one or more of the `LINEADDRCAPFLAGS_` constants.

`dwCallFeatures`

Specifies the switching capabilities or features available for all calls on this address using the `LINECALLFEATURE_` constants. This member represents the call-related features that may possibly be available on an address (static availability as opposed to dynamic availability). Invoking a supported feature requires the call to be in the proper state and the underlying line device to be opened in a compatible mode. A zero in a bit position indicates that the corresponding feature is never available. A one indicates that the corresponding feature may be available if the application has the right privileges to the call and the call is in the appropriate state for the operation to be meaningful. This member allows an application to discover which call features can be (and which can never be) supported by the address.

`dwRemoveFromConfCaps`

Specifies the address’s capabilities for removing calls from a conference call. This member uses one of the `LINEREMOVEFROMCONF_` constants.

`dwRemoveFromConfState`

Uses the `LINECALLSTATE_` constants to specify the state of the call after it has been removed from a conference call.

`dwTransferModes`

Specifies the address’s capabilities for resolving transfer requests. This member uses one of the `LINETRANSFERMODE_` constants.

`dwParkModes`

Specifies the different call park modes available at this address. This member uses one of the `LINEPARKMODE_` constants.

`dwForwardModes`

Specifies the different modes of forwarding available for this address. This member uses the `LINEFORWARDMODE_` constants.

`dwMaxForwardEntries`

Specifies the maximum number of entries that can be passed to `lineForward` in the `lpForwardList` parameter.

`dwMaxSpecificEntries`

Specifies the maximum number of entries in the `lpForwardList` parameter passed to `lineForward` that can contain forwarding instructions based on a specific caller ID (selective call forwarding). This member is zero if selective call forwarding is not supported.

`dwMinFwdNumRings`

Specifies the minimum number of rings that can be set to determine when a call is officially considered “no answer.”

`dwMaxFwdNumRings`

Specifies the maximum number of rings that can be set to determine when a call is officially considered “no answer.” If this number of rings cannot be set, then `dwMinFwdNumRings` and `dwMaxNumRings` are equal.

`dwMaxCallCompletions`

Specifies the maximum number of concurrent call completion requests that can be outstanding on this line device. Zero implies that call completion is not available.

`dwCallCompletionConds`

Specifies the different call conditions under which call completion can be requested. This member uses one or more of the `LINECALLCOMPLCOND_` constants.

`dwCallCompletionModes`

Specifies the way in which the call can be completed. This member uses one of the `LINECALLCOMPLMODE_` constants.

`dwNumCompletionMessages`

Specifies the number of call completion messages that can be selected from when using the `LINECALLCOMPLMODE_MESSAGE` option. Individual messages are identified by values in the range zero through one less than `dwNumCompletionMessages`.

`dwCompletionMsgTextEntrySize`

Specifies the size, in bytes, of each of the call completion text descriptions pointed at by `dwCompletionMsgTextSize` and `dwCompletionMsgTextOffset`.

`dwCompletionMsgTextSize`

`dwCompletionMsgTextOffset`

The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized field containing descriptive text about each of the call completion messages. Each message is `dwCompletionMsgTextEntrySize` bytes long. The string format of these textual descriptions is indicated by `dwStringFormat` in the line's device capabilities.

`dwAddressFeatures`

Specifies the features available for this address using the `LINEADDRFEATURE_` constants.

Invoking a supported feature requires the address to be in the proper state and the underlying line device to be opened in a compatible mode. A zero in a bit position indicates that the corresponding feature is never available. A one indicates that the corresponding feature may be available if the address is in the appropriate state for the operation to be meaningful. This member allows an application to discover which address features can be (and which can never be) supported by the address.

`dwPredictiveAutoTransferStates`

The call state or states upon which a call made by a predictive dialer can be set to automatically transfer the call to another address; one or more of the `LINECALLSTATE_` constants. The value 0 indicates automatic transfer based on call state is unavailable.

`dwNumCallTreatments`

The number of entries in the array of LINECALLTREATMENTENTRY structures delimited by dwCallTreatmentListSize and dwCallTreatmentListOffset.

dwCallTreatmentListSize

dwCallTreatmentListOffset

The total size, in bytes, and offset from the beginning of LINEADDRESSCAPS of an array of LINECALLTREATMENTENTRY structures, indicating the call treatments supported on the address (that can be selected using lineSetCallTreatment). The value is dwNumCallTreatments times SIZEOF(LINECALLTREATMENTENTRY).

dwDeviceClassesSize

dwDeviceClassesOffset

Length in bytes and offset from the beginning of LINEADDRESSCAPS of a string consisting of the device class identifiers supported on this address for use with lineGetID, separated by NULLs; the last class identifier is followed by two NULLs.

dwMaxCallDataSize

The maximum number of bytes that an application can set in LINECALLINFO using lineSetCallData.

dwCallFeatures2

Specifies additional switching capabilities or features available for all calls on this address using the LINECALLFEATURE2_ constants. It is an extension of the dwCallFeatures member.

dwMaxNoAnswerTimeout

The maximum value in seconds that can be set in the dwNoAnswerTimeout member in LINECALLPARAMS when making a call. A value of 0 indicates that automatic abandonment of unanswered calls is not supported by the service provider, or that the timeout value is not adjustable by applications.

dwConnectedModes

Specifies the LINECONNECTEDMODE_ values that can appear in the dwCallStateMode member of LINECALLSTATUS and in LINE_CALLSTATE messages for calls on this address.

dwOfferingModes

Specifies the LINEOFFERINGMODE_ values that can appear in the dwCallStateMode member of LINECALLSTATUS and in LINE_CALLSTATE messages for calls on this address.

dwAvailableMediaModes

Indicates the media types (modes) that can be invoked on new calls created on this address, when the dwAddressFeatures member indicates that new calls are possible. If this number is zero, it indicates that the service provider either does not know or cannot indicate which media types are available, in which case any or all of the media types indicated in the dwMediaModes member in LINEDEVCAPS may be available.

LINEADDRESSSTATUS

Description

The LINEADDRESSSTATUS structure describes the current status of an address. The lineGetAddressStatus function and the TSPI_lineGetAddressStatus function return the LINEADDRESSSTATUS structure.

Function Details

```
typedef struct lineaddressstatus_tag {
    DWORD   dwTotalSize;
    DWORD   dwNeededSize;
    DWORD   dwUsedSize;
    DWORD   dwNumInUse;
    DWORD   dwNumActiveCalls;
    DWORD   dwNumOnHoldCalls;
    DWORD   dwNumOnHoldPendCalls;
    DWORD   dwAddressFeatures;
    DWORD   dwNumRingsNoAnswer;
    DWORD   dwForwardNumEntries;
    DWORD   dwForwardSize;
    DWORD   dwForwardOffset;
    DWORD   dwTerminalModesSize;
    DWORD   dwTerminalModesOffset;
    DWORD   dwDevSpecificSize;
    DWORD   dwDevSpecificOffset;
} LINEADDRESSSTATUS, FAR *LPLINEADDRESSSTATUS;
```

Parameters

dwTotalSize

The total size, in bytes, 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.

dwNumInUse

Specifies the number of stations that are currently using the address.

dwNumActiveCalls

The number of calls on the address that are in call states other than idle, on hold, on hold pending transfer, and on hold pending conference.

dwNumOnHoldCalls

The number of calls on the address in the on-hold state.

dwNumOnHoldPendCalls

The number of calls on the address in the on-hold-pending-transfer or on-hold-pending-conference state.

`dwAddressFeatures`

Specifies the address-related API functions that can be invoked on the address in its current state. This member uses one or more of the `LINEADDRFEATURE_` constants.

`dwNumRingsNoAnswer`

The number of rings set for this address before an unanswered call is considered as no answer.

`dwForwardNumEntries`

The number of entries in the array referred to by `dwForwardSize` and `dwForwardOffset`.

`dwForwardSize`

`dwForwardOffset`

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 is an array of `dwForwardNumEntries` `LINEFORWARD` elements, which are relative to the beginning of the `LINEADDRESSSTATUS` structure. You must call upon the `dwCallerAddressOffset` and `dwDestAddressOffset` offset values in the `LINEFORWARD` field to derive the `dwForwardSize` and `dwForwardOffset` `LINEADDRESSSTATUS` offset values.

`dwTerminalModesSize`

`dwTerminalModesOffset`

The size and the offset, in bytes, from the beginning of the device field data structure containing an array with `DWORD`-sized entries using one or more of the `LINETERMMODE_` constants.

Terminal identifiers access the range from zero to the `dwNumTerminals` value minus one. Each entry in the array specifies the current terminal modes for the corresponding terminal set using a `lineSetTerminal` function to determine the address.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized device-specific field.

LINEAPPINFO

Description

The `LINEAPPINFO` structure contains information about the application that is currently running. The `LINEDEVSTATUS` structure can contain an array of `LINEAPPINFO` structures.

```
.Function 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;

```

Parameters

`dwMachineNameSize`

`dwMachineNameOffset`

Size, in bytes, and offset from the beginning of `LINEDEVSTATUS` of a string specifying the name of the computer on which the application is executing.

`dwUserNameSize`

`dwUserNameOffset`

Size, in bytes, and offset from the beginning of `LINEDEVSTATUS` of a string specifying the user name under whose account the application is running.

`dwModuleFilenameSize`

`dwModuleFilenameOffset`

Size, in bytes, and offset from the beginning of `LINEDEVSTATUS` of a string specifying the module filename of the application. This string can be used in a call to `lineHandoff` to perform a directed hand-off to the application.

`dwFriendlyNameSize`

`dwFriendlyNameOffset`

Size, in bytes, and offset from the beginning of `LINEDEVSTATUS` of the string provided by the application 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 when it opened the line `dwPrivileges` did not include `LINECALLPRIVILEGE_OWNER`.

`dwAddressID`

If the line handle was opened using `LINEOPENOPTION_SINGLEADDRESS`, contains the address identifier specified; set to `0xFFFFFFFF` if the single address option was not used.

An address identifier is permanently associated with an address; the identifier remains constant across operating system upgrades.

LINECALLINFO

Description

The `LINECALLINFO` structure contains information about a call. This information remains relatively fixed for the duration of the call. Multiple functions use `LINECALLINFO`. The structure is returned by the `lineGetCallInfo` function and the `TSPI_lineGetCallInfo` function. If a part of the structure does change, then a `LINE_CALLINFO` message is sent to the application indicating which information item has changed.

Dynamically changing information about a call, such as call progress status, is available in the LINECALLSTATUS structure, returned by a call to the lineGetCallStatus function.

Function Details

```
typedef struct linecallinfo_tag {
    DWORD   dwTotalSize;
    DWORD   dwNeededSize;
    DWORD   dwUsedSize;
    HLINE   hLine;
    DWORD   dwLineDeviceID;
    DWORD   dwAddressID;
    DWORD   dwBearerMode;
    DWORD   dwRate;
    DWORD   dwMediaMode;
    DWORD   dwAppSpecific;
    DWORD   dwCallID;
    DWORD   dwRelatedCallID;
    DWORD   dwCallParamFlags;
    DWORD   dwCallStates;
    DWORD   dwMonitorDigitModes;
    DWORD   dwMonitorMediaModes;
    LINEDIALPARAMS DialParams;
    DWORD   dwOrigin;
    DWORD   dwReason;
    DWORD   dwCompletionID;
    DWORD   dwNumOwners;
    DWORD   dwNumMonitors;
    DWORD   dwCountryCode;
    DWORD   dwTrunk;
    DWORD   dwCallerIDFlags;
    DWORD   dwCallerIDSize;
    DWORD   dwCallerIDOffset;
    DWORD   dwCallerIDNameSize;
    DWORD   dwCallerIDNameOffset;
    DWORD   dwCalledIDFlags;
    DWORD   dwCalledIDSize;
    DWORD   dwCalledIDOffset;
    DWORD   dwCalledIDNameSize;
    DWORD   dwCalledIDNameOffset;
    DWORD   dwConnectedIDFlags;
    DWORD   dwConnectedIDSize;
    DWORD   dwConnectedIDOffset;
    DWORD   dwConnectedIDNameSize;
    DWORD   dwConnectedIDNameOffset;
    DWORD   dwRedirectionIDFlags;
    DWORD   dwRedirectionIDSize;
    DWORD   dwRedirectionIDOffset;
    DWORD   dwRedirectionIDNameSize;
    DWORD   dwRedirectionIDNameOffset;
    DWORD   dwRedirectingIDFlags;
    DWORD   dwRedirectingIDSize;
    DWORD   dwRedirectingIDOffset;
    DWORD   dwRedirectingIDNameSize;
    DWORD   dwRedirectingIDNameOffset;
    DWORD   dwAppNameSize;
    DWORD   dwAppNameOffset;
    DWORD   dwDisplayableAddressSize;
    DWORD   dwDisplayableAddressOffset;
    DWORD   dwCalledPartySize;
    DWORD   dwCalledPartyOffset;
    DWORD   dwCommentSize;
}
```

```

    DWORD   dwCommentOffset;
    DWORD   dwDisplaySize;
    DWORD   dwDisplayOffset;
    DWORD   dwUserUserInfoSize;
    DWORD   dwUserUserInfoOffset;
    DWORD   dwHighLevelCompSize;
    DWORD   dwHighLevelCompOffset;
    DWORD   dwLowLevelCompSize;
    DWORD   dwLowLevelCompOffset;
    DWORD   dwChargingInfoSize;
    DWORD   dwChargingInfoOffset;
    DWORD   dwTerminalModesSize;
    DWORD   dwTerminalModesOffset;
    DWORD   dwDevSpecificSize;
    DWORD   dwDevSpecificOffset;
    DWORD   dwCallTreatment;
    DWORD   dwCallDataSize;
    DWORD   dwCallDataOffset;
    DWORD   dwSendingFlowspecSize;
    DWORD   dwSendingFlowspecOffset;
    DWORD   dwReceivingFlowspecSize;
    DWORD   dwReceivingFlowspecOffset;
} LINECALLINFO, FAR *LPLINECALLINFO;Parameters

```

Parameters

`dwTotalSize`

The total size, in bytes, 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.

`hLine`

The handle for the line device with which this call is associated.

`dwLineDeviceID`

The device identifier of the line device with which this call is associated.

`dwAddressID`

The address identifier of the address on the line on which this call exists. An address identifier is permanently associated with an address; the identifier remains constant across operating system upgrades.

`dwBearerMode`

The current bearer mode of the call. This member uses one of the `LINEBEARERMODE_` constants.

`dwRate`

The rate of the call's data stream in bps (bits per second).

`dwMediaMode`

Specifies the media type of the information stream currently on the call. This is the media type as determined by the owner of the call, which is not necessarily the same as that of the last `LINE_MONITORMEDIA` message. This member is not directly affected by the `LINE_MONITORMEDIA` messages. This member uses the `LINEMEDIAMODE_` constants.

`dwAppSpecific`

Not interpreted by the API implementation and service provider. It can be set by any owner application of this call with the `lineSetAppSpecific` function.

`dwCallID`

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.

`dwRelatedCallID`

Telephony environments that use the call ID often may find it necessary to relate one call to another. The `dwRelatedCallID` member may be used by the service provider for this purpose.

`dwCallParamFlags`

A collection of call-related parameters when the call is outgoing. These are the same call parameters specified in `lineMakeCall`, one or more of the `LINECALLPARAMFLAGS_` constants.

`dwCallStates`

The call states, one or more of the `LINECALLSTATE_` constants, for which the application can be notified on this call. The `dwCallStates` member is constant in `LINECALLINFO` and does not change depending on the call state.

`dwMonitorDigitModes`

The various digit modes, one or more of the `LINEDIGITMODE_` constants, for which monitoring is currently enabled.

`dwMonitorMediaModes`

The various media types for which monitoring is currently enabled, one or more of the `LINEMEDIAMODE_` constants.

`DialParams`

The dialing parameters currently in effect on the call, of type `LINEDIALPARAMS`. Unless these parameters are set by either `lineMakeCall` or `lineSetCallParams`, their values are the same as the defaults used in the `LINEDEVCAPS` structure.

`dwOrigin`

Identifies where the call originated, one of the `LINECALLORIGIN_` constants.

`dwReason`

The reason why the call occurred, one of the `LINECALLREASON_` constants.

`dwCompletionID`

The completion identifier for the incoming call if it is the result of a completion request that terminates. This identifier is meaningful only if `dwReason` is `LINECALLREASON_CALLCOMPLETION`.

`dwNumOwners`

The number of application modules with different call handles and owner privilege for the call.

`dwNumMonitors`

The number of application modules with different call handles and monitor privilege for the call.

`dwCountryCode`

The country code of the destination party. Zero if unknown.

`dwTrunk`

The number of the trunk over which the call is routed. This member is used for both incoming and outgoing calls. The `dwTrunk` member should be set to `0xFFFFFFFF` if it is unknown.

`dwCallerIDFlags`

Determines the validity and content of the caller, or originator, party identifier information. This member uses one of the `LINECALLPARTYID_` constants.

`dwCallerIDSize`

`dwCallerIDOffset`

The size, in bytes, of the variably sized field containing the caller party ID number information, and the offset, in bytes, from the beginning of this data structure.

`dwCallerIDNameSize`

`dwCallerIDNameOffset`

The size, in bytes, of the variably sized field containing the caller party ID name information, and the offset, in bytes, from the beginning of this data structure.

`dwCalledIDFlags`

Determines the validity and content of the called-party ID information. The called party corresponds to the originally addressed party. This member uses one of the `LINECALLPARTYID_` constants.

`dwCalledIDSize`

`dwCalledIDOffset`

The size, in bytes, of the variably sized field containing the called-party ID number information, and the offset, in bytes, from the beginning of this data structure.

`dwCalledIDNameSize`

`dwCalledIDNameOffset`

The size, in bytes, of the variably sized field containing the called-party ID name information, and the offset, in bytes, from the beginning of this data structure.

`dwConnectedIDFlags`

Determines the validity and content of the connected-party ID information. The connected party is the party that was actually connected to. This may be different from the called-party ID if the call was diverted. This member uses one of the `LINECALLPARTYID_` constants.

`dwConnectedIDSize`

`dwConnectedIDOffset`

The size, in bytes, of the variably sized field containing the connected party identifier number information, and the offset, in bytes, from the beginning of this data structure.

`dwConnectedIDNameSize`

`dwConnectedIDNameOffset`

The size, in bytes, of the variably sized field containing the connected party identifier name information, and the offset, in bytes, from the beginning of this data structure.

`dwRedirectionIDFlags`

Determines the validity and content of the redirection party identifier information. The redirection party identifies the address to which the session was redirected. This member uses one of the `LINECALLPARTYID_` constants.

`dwRedirectionIDSize`

`dwRedirectionIDOffset`

The size, in bytes, of the variably sized field containing the redirection party identifier number information, and the offset, in bytes, from the beginning of this data structure.

`dwRedirectionIDNameSize`

`dwRedirectionIDNameOffset`

The size, in bytes, of the variably sized field containing the redirection party identifier name information, and the offset, in bytes, from the beginning of this data structure.

`dwRedirectingIDFlags`

Determines the validity and content of the redirecting party identifier information. The redirecting party identifies the address which redirect the session. This member uses one of the `LINECALLPARTYID_` constants.

`dwRedirectingIDSize`

`dwRedirectingIDOffset`

The size, in bytes, of the variably sized field containing the redirecting party identifier number information, and the offset, in bytes, from the beginning of this data structure.

`dwRedirectingIDNameSize`

`dwRedirectingIDNameOffset`

The size, in bytes, of the variably sized field containing the redirecting party identifier name information, and the offset, in bytes, from the beginning of this data structure.

`dwAppNameSize`

`dwAppNameOffset`

The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized field holding the user-friendly application name of the application that first originated, accepted, or answered the call. This is the name that an application can specify in `lineInitializeEx`. If the application specifies no such name, then the application's module filename is used instead.

`dwDisplayableAddressSize`

`dwDisplayableAddressOffset`

The string is used for logging purposes. The information is obtained from LINECALLPARAMS for functions that initiate calls. The lineTranslateAddress function returns appropriate information to be placed in this field in the dwDisplayableAddressSize and dwDisplayableAddressOffset members of the LINETRANSLATEOUTPUT structure.

```
dwCalledPartySize
dwCalledPartyOffset
```

The size, in bytes, of the variably sized field holding a user-friendly description of the called party, and the offset, in bytes, from the beginning of this data structure. This information can be specified with lineMakeCall and can be optionally specified in the lpCallParams parameter whenever a new call is established. It is useful for call logging purposes.

```
dwCommentSize
dwCommentOffset
```

The size, in bytes, of the variably sized field holding a comment about the call provided by the application that originated the call using lineMakeCall, and the offset, in bytes, from the beginning of this data structure. This information can be optionally specified in the lpCallParams parameter whenever a new call is established.

```
dwDisplaySize
dwDisplayOffset
```

The size, in bytes, of the variably sized field holding raw display information, and the offset, in bytes, from the beginning of this data structure. Depending on the telephony environment, a service provider may extract functional information from this member pair for formatting and presentation most appropriate for this telephony configuration.

```
dwUserUserInfoSize
dwUserUserInfoOffset
```

The size, in bytes, of the variably sized field holding user-user information, and the offset, in bytes, from the beginning of this data structure. The protocol discriminator field for the user-user information, if used, appears as the first byte of the data pointed to by dwUserUserInfoOffset, and is accounted for in dwUserUserInfoSize.¹

```
dwHighLevelCompSize
dwHighLevelCompOffset
```

The size, in bytes, of the variably sized field holding high-level compatibility information, and the offset, in bytes, from the beginning of this data structure. The format of this information is specified by other standards (ISDN Q.931).

```
dwLowLevelCompSize
dwLowLevelCompOffset
```

The size, in bytes, of the variably sized field holding low-level compatibility information, and the offset, in bytes, from the beginning of this data structure. The format of this information is specified by other standards (ISDN Q.931).

```
dwChargingInfoSize
dwChargingInfoOffset
```

1. Cisco Unified CME TSP 2.1 does not support user-user information. This should be set to NULL.

The size, in bytes, of the variably sized field holding charging information, and the offset, in bytes, from the beginning of this data structure. The format of this information is specified by other standards (ISDN Q.931).

`dwTerminalModesSize`

`dwTerminalModesOffset`

The size, in bytes, of the variably sized device field containing an array with DWORD-sized entries, and the offset, in bytes, from the beginning of this data structure. Array entries are indexed by terminal identifiers, in the range from zero to one less than `dwNumTerminals`. Each entry in the array specifies the current terminal modes for the corresponding terminal set with the `lineSetTerminal` function for this call's media stream, as specified by one of the `LINETERMMODE_` constants.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, of the variably-sized field holding device-specific information, and the offset, in bytes, from the beginning of this data structure.

`dwCallTreatment`

The call treatment currently being applied on the call or that is applied when the call enters the next applicable state. Can be zero if call treatments are not supported.

`dwCallDataSize`

`dwCallDataOffset`

The size, in bytes, and offset from the beginning of `LINECALLINFO` of the application-specified call data.

`dwSendingFlowspecSize`

`dwSendingFlowspecOffset`

The total size, in bytes, and offset from the beginning of `LINECALLINFO` of a WinSock2 `FLOWSPEC` structure followed by WinSock2 provider-specific data, equivalent to what would have been stored in `SendingFlowspec.len` in a WinSock2 QOS structure. Specifies the quality of service current in effect in the sending direction on the call. The provider-specific portion following the `FLOWSPEC` structure must not contain pointers to other blocks of memory, because TAPI does not know how to marshal the data pointed to by the private pointer(s) and convey it to the application.

`dwReceivingFlowspecSize`

`dwReceivingFlowspecOffset`

As in `SendingFlowspecOffset`.

LINECALLLIST

Description

The `LINECALLLIST` structure describes a list of call handles. A structure of this type is returned by the `lineGetNewCalls` and `lineGetConfRelatedCalls` functions.

Function Details

```
typedef struct linecalllist_tag {
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    DWORD dwCallsNumEntries;
    DWORD dwCallsSize;
    DWORD dwCallsOffset;
} LINECALLLIST, FAR *LPLINECALLLIST;
```

Parameters

`dwTotalSize`

The total size, in bytes, 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.

`dwCallsSize`

`dwCallsOffset`

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

Description

The `LINECALLPARAMS` structure describes parameters supplied when making calls using the `lineMakeCall` and `TSPI_lineMakeCall` functions. The `LINECALLPARAMS` structure is also used as a parameter in other operations, such as the `lineOpen` function.

The comments to the right of the syntax block indicate the default values used when this structure is not provided to `lineMakeCall`.

Function Details

```
typedef struct linecallparams_tag { // Defaults:
    DWORD dwTotalSize;           // -----
    DWORD dwBearerMode;          // voice
    DWORD dwMinRate;              // (3.1kHz)
    DWORD dwMaxRate;              // (3.1kHz)
    DWORD dwMediaMode;           // interactiveVoice
    DWORD dwCallParamFlags;      // 0
    DWORD dwAddressMode;         // addressID
    DWORD dwAddressID;           // (any available)
```

```

    LINECALLPARAMS DialParams; // (0, 0, 0, 0)
    DWORD   dwOrigAddressSize;      // 0
    DWORD   dwOrigAddressOffset;
    DWORD   dwDisplayableAddressSize; // 0
    DWORD   dwDisplayableAddressOffset;
    DWORD   dwCalledPartySize;      // 0
    DWORD   dwCalledPartyOffset;
    DWORD   dwCommentSize;          // 0
    DWORD   dwCommentOffset;
    DWORD   dwUserUserInfoSize;     // 0
    DWORD   dwUserUserInfoOffset;
    DWORD   dwHighLevelCompSize;    // 0
    DWORD   dwHighLevelCompOffset;
    DWORD   dwLowLevelCompSize;     // 0
    DWORD   dwLowLevelCompOffset;
    DWORD   dwDevSpecificSize;      // 0
    DWORD   dwDevSpecificOffset;
; //TAPI Version 2.1
    DWORD   dwPredictiveAutoTransferStates;
    DWORD   dwTargetAddressSize;
    DWORD   dwTargetAddressOffset;
    DWORD   dwSendingFlowspecSize;
    DWORD   dwSendingFlowspecOffset;
    DWORD   dwReceivingFlowspecSize;
    DWORD   dwReceivingFlowspecOffset;
    DWORD   dwDeviceClassSize;
    DWORD   dwDeviceClassOffset;
    DWORD   dwDeviceConfigSize;
    DWORD   dwDeviceConfigOffset;
    DWORD   dwCallDataSize;
    DWORD   dwCallDataOffset;
    DWORD   dwNoAnswerTimeout;
    DWORD   dwCallingPartyIDSize;
    DWORD   dwCallingPartyIDOffset;
; //TAPI Version 3.0 and higher
    DWORD   dwAddressType;
} LINECALLPARAMS, FAR *LPLINECALLPARAMS;

```

Parameters

`dwTotalSize`

The total size, in bytes, allocated to this data structure. This size should be big enough to hold all the fixed and variably sized portions of this data structure.

`dwBearerMode`

The bearer mode for the call. This member uses one of the `LINEBEARERMODE_` constants.

If `dwBearerMode` is zero, the default value is `LINEBEARERMODE_VOICE`.

`dwMinRate`

`dwMaxRate`

The data rate range requested for the call's data stream in bps (bits per second). When making a call, the service provider attempts to provide the highest available rate in the requested range. If a specific data rate is required, both the minimum and maximum should be set to that value. If an application works best with one rate but is able to degrade to lower rates, the application should specify these as the maximum and minimum rates, respectively. If `dwMaxRate` is zero, the default value is as specified by the `dwMaxRate` member of the `LINEDEVCAPS` structure. This is the maximum rate supported by the device.

`dwMediaMode`

The expected media type of the call. This member uses one of the `LINEMEDIAMODE_` constants. If `dwMediaMode` is zero, the default value is `LINEMEDIAMODE_INTERACTIVEVOICE`.

`dwCallParamFlags`

These flags specify a collection of Boolean call-setup parameters. This member uses one or more of the `LINECALLPARAMFLAGS_` constants.

`dwAddressMode`

The mode by which the originating address is specified. The `dwAddressMode` member cannot be `LINEADDRESSMODE_ADDRESSID` for the `lineOpen` function call. This member uses one of the `LINEADDRESSMODE_` constants.

`dwAddressID`

The address identifier of the originating address if `dwAddressMode` is set to `LINEADDRESSMODE_ADDRESSID`. An address identifier is permanently associated with an address; the identifier remains constant across operating system upgrades.

`DialParams`

Dial parameters to be used on this call, of type `LINEDIALPARAMS`. When a value of 0 is specified for this field, the default value for the field is used as indicated in the `DefaultDialParams` member of the `LINEDEVCAPS` structure. If a nonzero value is specified for a field that is outside the range specified by the corresponding fields in `MinDialParams` and `MaxDialParams` in the `LINEDEVCAPS` structure, the nearest value within the valid range is used instead.

`dwOrigAddressSize`

`dwOrigAddressOffset`

The size, in bytes, of the variably sized field holding the originating address, and the offset, in bytes, from the beginning of this data structure. The format of this address is dependent on the `dwAddressMode` member.

`dwDisplayableAddressSize`

`dwDisplayableAddressOffset`

The displayable string is used for logging purposes. The content of these members is recorded in the `dwDisplayableAddressOffset` and `dwDisplayableAddressSize` members of the call's `LINECALLINFO` message. The `lineTranslateAddress` function returns appropriate information to be placed in this field in the `dwDisplayableAddressSize` and `dwDisplayableAddressOffset` members of the `LINETRANSLATEOUTPUT` structure.

`dwCalledPartySize`

`dwCalledPartyOffset`

The size, in bytes, of the variably sized field holding called-party information, and the offset, in bytes, from the beginning of this data structure. This information can be specified by the application that makes the call and is made available in the call's information structure for logging purposes. The format of this field is that of `dwStringFormat`, as specified in `LINEDEVCAPS`.

`dwCommentSize`

`dwCommentOffset`

The size, in bytes, of the variably sized field holding comments about the call, and the offset, in bytes, from the beginning of this data structure. This information can be specified by the application that makes the call and is made available in the call's information structure for logging purposes. The format of this field is that of `dwStringFormat`, as specified in `LINEDEVCAPS`.

`dwUserUserInfoSize`

`dwUserUserInfoOffset`

The size, in bytes, of the variably sized field holding user-user information, and the offset, in bytes, from the beginning of this data structure. The protocol discriminator field for the user-user information, if required, should appear as the first byte of the data pointed to by `dwUserUserInfoOffset`, and must be accounted for in `dwUserUserInfoSize`.¹

`dwHighLevelCompSize`

`dwHighLevelCompOffset`

The size, in bytes, of the variably sized field holding high-level compatibility information, and the offset, in bytes, from the beginning of this data structure.

`dwLowLevelCompSize`

`dwLowLevelCompOffset`

The size, in bytes, of the variably sized field holding low-level compatibility information, and the offset, in bytes, from the beginning of this data structure.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, of the variably sized field holding device-specific information, and the offset, in bytes, from the beginning of this data structure.

`dwPredictiveAutoTransferStates`

The `LINECALLSTATE_` constants, entry into which causes the call to be blind-transferred to the specified target address. Set to zero if automatic transfer is not desired.

`dwTargetAddressSize`

`dwTargetAddressOffset`

The size, in bytes, and offset from the beginning of `LINECALLPARAMS` of a string specifying the target dialable address (not `dwAddressID`); used in the case of certain automatic actions. In the case of predictive dialing, specifies the address to which the call should be automatically transferred. This is essentially the same string that would be passed to `lineBlindTransfer` if automatic transfer were not being used. Set to zero if automatic transfer is not desired. In the case of a No Hold Conference, specifies the address that should be conferenced to the call. In the case of a One Step Transfer, specifies the address to dial on the consultation call.

`dwSendingFlowspecSize`

`dwSendingFlowspecOffset`

The total size, in bytes, and offset from the beginning of `LINECALLPARAMS` of a `WinSock2 FLOWSPEC` structure followed by `WinSock2` provider-specific data, equivalent to what would have been stored in `SendingFlowspec.len` in a `WinSock2 QOS` structure. Specifies the quality of service

1. Cisco Unified CME TSP 2.1 does not support user-user information. This should be set to `NULL`.

desired in the sending direction on the call. The provider-specific portion following the FLOWSPEC structure must not contain pointers to other blocks of memory, because TAPI does not know how to marshal the data pointed to by the private pointer(s) and convey it to the application.

`dwReceivingFlowspecSize`

`dwReceivingFlowspecOffset`

The total size, in bytes, and offset from the beginning of LINECALLPARAMS of a WinSock2 FLOWSPEC structure followed by WinSock2 provider-specific data, equivalent to what would have been stored in ReceivingFlowspec.len in a WinSock2 QOS structure. Specifies the quality of service desired in the receiving direction on the call. The provider-specific portion following the FLOWSPEC structure must not contain pointers to other blocks of memory, because TAPI does not know how to marshal the data pointed to by the private pointer(s) and convey it to the application.

`dwDeviceClassSize`

`dwDeviceClassOffset`

The size, in bytes, and offset from the beginning of LINECALLPARAMS of a null-terminated string (the size includes the NULL) that indicates the device class of the device whose configuration is specified in DeviceConfig. Valid device class strings are the same as those specified for the lineGetID function.

`dwDeviceConfigSize`

`dwDeviceConfigOffset`

The number of bytes and offset from the beginning of LINECALLPARAMS of the opaque configuration data structure pointed to by dwDevConfigOffset. This value is returned in the dwStringSize member in the VARSTRING structure returned by lineGetDevConfig. If the size is zero, the default device configuration is used. This allows the application to set the device configuration before the call is initiated.

`dwCallDataSize`

`dwCallDataOffset`

The size, in bytes, and offset from the beginning of LINECALLPARAMS of the application-specified call data to be initially attached to the call.

`dwNoAnswerTimeout`

The number of seconds, after the completion of dialing, that the call should be allowed to wait in the PROCEEDING or RINGBACK states, before it is automatically abandoned by the service provider with a LINECALLSTATE_DISCONNECTED and LINEDISCONNECTMODE_NOANSWER. A value of 0 indicates that the application does not desire automatic call abandonment.

`dwCallingPartyIDSize`

`dwCallingPartyIDOffset`

The size, in bytes, and offset from the beginning of LINECALLPARAMS of a null-terminated string (the size includes the NULL) that specifies the identity of the party placing the call. If the content of the identifier is acceptable and a path is available, the service provider passes the identifier along to the called party to indicate the identity of the calling party.

`dwAddressType`

The address type used for the call. This member of the structure is available only if the negotiated TAPI version is 3.0 or higher.

LINECALLSTATUS

Description

The LINECALLSTATUS structure describes the current status of a call. The information in this structure depends on the device capabilities of the address, the ownership of the call by the invoking application, and the current state of the call being queried. The lineGetCallStatus and TSPI_lineGetCallStatus functions return the LINECALLSTATUS structure.

Function Details

```
typedef struct linecallstatus_tag {
    DWORD   dwTotalSize;
    DWORD   dwNeededSize;
    DWORD   dwUsedSize;
    DWORD   dwCallState;
    DWORD   dwCallStateMode;
    DWORD   dwCallPrivilege;
    DWORD   dwCallFeatures;
    DWORD   dwDevSpecificSize;
    DWORD   dwDevSpecificOffset;
    DWORD   dwCallFeatures2;
    SYSTEMTIME tStateEntryTime;
} LINECALLSTATUS, FAR *LPLINECALLSTATUS;
```

Parameters

dwTotalSize

The total size, in bytes, 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.

dwCallState

Specifies the current call state of the call using one of the LINECALLSTATE_ constants.

dwCallStateMode

The interpretation of the dwCallStateMode member is call-state-dependent. In many cases, the value will be zero.

dwCallPrivilege

The application's privilege for this call. This member uses one or more of the LINECALLPRIVILEGE_ constants.

dwCallFeatures

These flags indicate the telephony API functions that can be invoked on the call, given the availability of the feature in the device capabilities, the current call state, and call ownership of the invoking application. A zero indicates the corresponding feature cannot be invoked by the application on the call in its current state; a one indicates the feature can be invoked. This member uses LINECALLFEATURE_ constants.

```
dwDevSpecificSize
dwDevSpecificOffset
```

The size, in bytes, of the variably sized device-specific field, and the offset, in bytes, from the beginning of this data structure.

```
dwCallFeatures2
```

Indicates additional functions can be invoked on the call, given the availability of the feature in the device capabilities, the current call state, and call ownership of the invoking application. An extension of the dwCallFeatures member. This member uses LINECALLFEATURE2_ constants.

```
tStateEntryTime
```

The Coordinated Universal Time at which the current call state was entered.

LINEDEVCAPS

Description

The LINEDEVCAPS structure describes the capabilities of a line device. The lineGetDevCaps function and the TSPI_lineGetDevCaps function return the LINEDEVCAPS structure.

Function Details

```
typedef struct linedevcaps_tag {
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    DWORD dwProviderInfoSize;
    DWORD dwProviderInfoOffset;
    DWORD dwSwitchInfoSize;
    DWORD dwSwitchInfoOffset;
    DWORD dwPermanentLineID;
    DWORD dwLineNameSize;
    DWORD dwLineNameOffset;
    DWORD dwStringFormat;
    DWORD dwAddressModes;
    DWORD dwNumAddresses;
    DWORD dwBearerModes;
    DWORD dwMaxRate;
    DWORD dwMediaModes;
    DWORD dwGenerateToneModes;
    DWORD dwGenerateToneMaxNumFreq;
    DWORD dwGenerateDigitModes;
    DWORD dwMonitorToneMaxNumFreq;
    DWORD dwMonitorToneMaxNumEntries;
    DWORD dwMonitorDigitModes;
    DWORD dwGatherDigitsMinTimeout;
    DWORD dwGatherDigitsMaxTimeout;
    DWORD dwMedCtlDigitMaxListSize;
```

```

DWORD dwMedCtlMediaMaxListSize;
DWORD dwMedCtlToneMaxListSize;
DWORD dwMedCtlCallStateMaxListSize;
DWORD dwDevCapFlags;
DWORD dwMaxNumActiveCalls;
DWORD dwAnswerMode;
DWORD dwRingModes;
DWORD dwLineStates;
DWORD dwUIIAcceptSize;
DWORD dwUIIAnswerSize;
DWORD dwUIIMakeCallSize;
DWORD dwUIDropSize;
DWORD dwUISendUserUserInfoSize;
DWORD dwUICallInfoSize;
LINEDIALPARAMS MinDialParams;
LINEDIALPARAMS MaxDialParams;
LINEDIALPARAMS DefaultDialParams;
DWORD dwNumTerminals;
DWORD dwTerminalCapsSize;
DWORD dwTerminalCapsOffset;
DWORD dwTerminalTextEntrySize;
DWORD dwTerminalTextSize;
DWORD dwTerminalTextOffset;
DWORD dwDevSpecificSize;
DWORD dwDevSpecificOffset;
DWORD dwLineFeatures;
DWORD dwSettableDevStatus;
DWORD dwDeviceClassesSize;
DWORD dwDeviceClassesOffset;
//TAPI Version 2.2
GUID PermanentLineGuid;
//TAPI Version 3.0
DWORD dwAddressTypes;
GUID ProtocolGuid;
DWORD dwAvailableTracking;
} LINEDEVCAPS, FAR *LPLINEDEVCAPS;

```

Parameters

dwTotalSize

The total size, in bytes, 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.

dwProviderInfoSize

dwProviderInfoOffset

The size, in bytes, of the variably sized field containing service provider information, and the offset, in bytes, from the beginning of this data structure. The dwProviderInfoSize/Offset member is intended to provide information about the provider hardware and/or software, such as the vendor name and version numbers of hardware and software. This information can be useful when a user needs to call customer service with problems regarding the provider.

dwSwitchInfoSize

`dwSwitchInfoOffset`

The size, in bytes, of the variably sized device field containing switch information, and the offset, in bytes, from the beginning of this data structure. The `dwSwitchInfoSize/Offset` member is intended to provide information about the switch to which the line device is connected, such as the switch manufacturer, the model name, the software version, and so on. This information can be useful when a user needs to call customer service with problems regarding the switch.

`dwPermanentLineID`

The permanent DWORD identifier by which the line device is known in the system's configuration. This permanent name, as opposed to `dwDevice ID`, does not change as lines are added or removed from the system and persists through operating system upgrades. It can therefore be used 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`

The size, in bytes, of the variably sized device field containing a user-configurable name for this line device, and the offset, in bytes, from the beginning of this data structure. This name can be configured by the user when configuring the line device's service provider, and is provided for the user's convenience.

`dwStringFormat`

The string format used with this line device. This member uses one of the `STRINGFORMAT_` constants.

`dwAddressModes`

The mode by which the originating address is specified. This member uses the `LINEADDRESSMODE_` constants.

`dwNumAddresses`

The number of addresses associated with this line device. Individual addresses are referred to by address identifiers. Address identifiers range from zero to one less than the value indicated by `dwNumAddresses`.

`dwBearerModes`

Flag array that indicates the different bearer modes that the address is able to support. This member uses one or more of the `LINEBEARERMODE_` constants.

`dwMaxRate`

Contains the maximum data rate, in bits per second, for information exchange over the call.

`dwMediaModes`

Flag array that indicates the different media types the address is able to support. This member uses one or more of the `LINEMEDIAMODE_` constants.

`dwGenerateToneModes`

The different kinds of tones that can be generated on this line. This member uses one or more of the `LINETONEMODE_` constants.

`dwGenerateToneMaxNumFreq`

Specifies the maximum number of frequencies you can configure for the `lineGenerateTone` setting in the `LINEGENERATETONE` data structure. A value of 0 indicates that tone generation is not available.

`dwGenerateDigitModes`

Specifies the digit modes than can be generated on this line. This member uses one or more of the `LINEDIGITMODE_` constants.

`dwMonitorToneMaxNumFreq`

Specifies the maximum number of frequencies you can configure for the `lineMonitorTones` setting in the the `LINEMONITORTONE` data structure. A value of 0 indicates that tone monitor is not available.

`dwMonitorToneMaxNumEntries`

Contains the maximum number of entries that can be specified in a tone list to `lineMonitorTones`.

`dwMonitorDigitModes`

Specifies the digit modes than can be detected on this line. This member uses one or more of the `LINEDIGITMODE_` constants.

`dwGatherDigitsMinTimeout`

`dwGatherDigitsMaxTimeout`

These members contain the minimum and maximum values, in milliseconds, that can be specified for both the first digit and inter-digit timeout values used by `lineGatherDigits`. If both these members are zero, timeouts are not supported.

`dwMedCtlDigitMaxListSize`

`dwMedCtlMediaMaxListSize`

`dwMedCtlToneMaxListSize`

`dwMedCtlCallStateMaxListSize`

These members contain the maximum number of entries that can be specified in the digit list, the media list, the tone list, and the call state list parameters of `lineSetMediaControl` respectively.

`dwDevCapFlags`

Specifies various Boolean device capabilities. This member uses one or more of the `LINEDEVCAPFLAGS_` constants.

`dwMaxNumActiveCalls`

Provides the maximum number of (minimum bandwidth) calls that can be active connected on the line at any one time. The actual number of active calls may be lower if higher bandwidth calls have been established on the line.

`dwAnswerMode`

Specifies the effect on the active call when answering another offering call on a line device. This member uses one of the `LINEANSWERMODE_` constants.

`dwRingModes`

Contains the number of different ring modes that can be reported in the `LINE_LINEDEVSTATE` message with the ringing indication. Different ring modes range from one to `dwRingModes`. Zero indicates no ring.

`dwLineStates`

Specifies the different line status components for which the application may be notified in a `LINE_LINEDEVSTATE` message on this line. This member uses one or more of the `LINEDEVSTATE_` constants.

`dwUUIAcceptSize`

Specifies the maximum size of user-user information that can be sent during a call accept.¹

`dwUUIAnswerSize`

Specifies the maximum size of user-user information that can be sent during a call answer.¹

`dwUUIMakeCallSize`

Specifies the maximum size of user-user information that can be sent during a make call.¹

`dwUUIDropSize`

Specifies the maximum size of user-user information that can be sent during a call drop.¹

`dwUUISendUserUserInfoSize`

Specifies the maximum size of user-user information that can be sent separately any time during a call with `lineSendUserUserInfo`.¹

`dwUUICallInfoSize`

Specifies the maximum size of user-user information that can be received in the `LINECALLINFO` structure.¹

`MinDialParams`

`MaxDialParams`

These members contain the minimum and maximum values, in milliseconds, for the dial parameters that can be set for calls on this line. Dialing parameters can be set to values in this range. The granularity of the actual settings is service provider-specific.

`DefaultDialParams`

Contains the default dial parameters used for calls on this line. These parameter values can be overridden on a per-call basis.

`dwNumTerminals`

The number of terminals that can be set for this line device, its addresses, or its calls. Individual terminals are referred to by terminal IDs and range from zero to one less than the value indicated by `dwNumTerminals`.

`dwTerminalCapsSize`

`dwTerminalCapsOffset`

The size, in bytes, and the offset, in bytes, from the beginning of this data structure of the variably sized device field containing an array with entries of type `LINETERMCAPS`. This array is indexed by terminal IDs, in the range from zero to `dwNumTerminals` minus one. Each entry in the array specifies the terminal device capabilities of the corresponding terminal.

`dwTerminalTextEntrySize`

The size, in bytes, of each of the terminal text descriptions pointed at by `dwTerminalTextSize/Offset`.

`dwTerminalTextSize`

1. Cisco Unified CME TSP 2.1 does not support user-user information. This should be set to `NULL`.

`dwTerminalTextOffset`

The size, in bytes, of the variably sized field containing descriptive text about each of the line's available terminals, and the offset, in bytes, from the beginning of this data structure. Each message is `dwTerminalTextEntrySize` bytes long. The string format of these textual descriptions is indicated by `dwStringFormat` in the line's device capabilities.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, of the variably sized device-specific field, and the offset, in bytes, from the beginning of this data structure.

`dwLineFeatures`

Specifies the features available for this line using the `LINEFEATURE_` constants. Invoking a supported feature requires the line to be in the proper state and the underlying line device to be opened in a compatible mode. A zero in a bit position indicates that the corresponding feature is never available. A one indicates that the corresponding feature may be available if the line is in the appropriate state for the operation to be meaningful. This member allows an application to discover which line features can be (and which can never be) supported by the device.

`dwSettableDevStatus`

The `LINEDEVSTATUS` values that can be modified using `lineSetLineDevStatus`.

`dwDeviceClassesSize`

`dwDeviceClassesOffset`

Length, in bytes, and offset from the beginning of `LINEDEVCAPS` of a string consisting of the device class identifiers supported on one or more addresses on this line for use with `lineGetID`, separated by NULLs; the last identifier in the list is followed by two NULLs.

`PermanentLineGuid`

The GUID permanently associated with the line device.

`dwAddressTypes`

The address type used for the call. This member of the structure is available only if the negotiated TAPI version is 3.0 or higher.

`ProtocolGuid`

The current TAPI Protocol. This member of the structure is available only if the negotiated TAPI version is 3.0 or higher. The protocols are declared in `tapi3.h`.

`dwAvailableTracking`

Available tracking, as represented by a `LINECALLHUBTRACKING` constant. This member of the structure is available only if the negotiated TAPI version is 3.0 or higher.

LINEDEVSTATUS

Description

The LINEDEVSTATUS structure describes the current status of a line device. The lineGetLineDevStatus function and the TSPI_lineGetLineDevStatus function return the LINEDEVSTATUS structure.

Function Details

```
typedef struct linedevstatus_tag {
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    DWORD dwNumOpens;
    DWORD dwOpenMediaModes;
    DWORD dwNumActiveCalls;
    DWORD dwNumOnHoldCalls;
    DWORD dwNumOnHoldPendCalls;
    DWORD dwLineFeatures;
    DWORD dwNumCallCompletions;
    DWORD dwRingMode;
    DWORD dwSignalLevel;
    DWORD dwBatteryLevel;
    DWORD dwRoamMode;
    DWORD dwDevStatusFlags;
    DWORD dwTerminalModesSize;
    DWORD dwTerminalModesOffset;
    DWORD dwDevSpecificSize;
    DWORD dwDevSpecificOffset;
    DWORD dwAvailableMediaModes;
    DWORD dwAppInfoSize;
    DWORD dwAppInfoOffset;
} LINEDEVSTATUS, FAR *LPLINEDEVSTATUS;
```

Parameters

`dwTotalSize`

The total size, in bytes, 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.

`dwNumOpens`

The number of active opens on the line device.

`dwOpenMediaModes`

Bit array that indicates for which media types the line device is currently open.

`dwNumActiveCalls`

The number of calls on the line in call states other than idle, onhold, onholdpendingtransfer, and onholdpendingconference.

`dwNumOnHoldCalls`

The number of calls on the line in the onhold state.

`dwNumOnHoldPendCalls`

The number of calls on the line in the onholdpendingtransfer or onholdpendingconference state.

`dwLineFeatures`

Specifies the line-related API functions that are currently available on this line. This member uses one or more of the `LINEFEATURE_` constants.

`dwNumCallCompletions`

The number of outstanding call completion requests on the line.

`dwRingMode`

The current ring mode on the line device.

`dwSignalLevel`

The current signal level of the connection on the line. This is a value in the range 0x00000000 (weakest signal) to 0x0000FFFF (strongest signal).

`dwBatteryLevel`

The current battery level of the line device hardware. This is a value in the range 0x00000000 (battery empty) to 0x0000FFFF (battery full).

`dwRoamMode`

The current roam mode of the line device. This member uses one of the `LINEROAMMODE_` constants.

`dwDevStatusFlags`

The status flags indicate information such as whether the device is locked. It consists of one or more members of `LINEDEVSTATUSFLAGS_` constants.

`dwTerminalModesSize`

`dwTerminalModesOffset`

The size, in bytes, of the variably sized device field containing an array with `DWORD`-sized entries, and the offset, in bytes, from the beginning of this data structure. This array is indexed by terminal IDs, in the range from zero to `dwNumTerminals` minus one. Each entry in the array specifies the current terminal modes for the corresponding terminal set using the `lineSetTerminal` function for this line. Each entry uses one or more of the `LINETERMMODE_` constants.

`dwDevSpecificSize`

`dwDevSpecificOffset`

The size, in bytes, of the variably sized device-specific field, and the offset, in bytes, from the beginning of this data structure.

`dwAvailableMediaModes`

Indicates the media types that can be invoked on new calls created on this line device, when the `dwLineFeatures` member indicates that new calls are possible. If this member is zero, it indicates that the service provider either does not know or cannot indicate which media types are available, in which case any or all of the media types indicated in the `dwMediaModes` member in `LINEDEVCAPS` may be available.

```
dwAppInfoSize
dwAppInfoOffset
```

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.

LINEFORWARD

Description

The LINEFORWARD structure describes an entry of the forwarding instructions. The LINEFORWARDLIST and the LINEADDRESSSTATUS structures can contain an array of LINEFORWARD structures.

Function Details

```
typedef struct lineforward_tag {
    DWORD dwForwardMode;
    DWORD dwCallerAddressSize;
    DWORD dwCallerAddressOffset;
    DWORD dwDestCountryCode;
    DWORD dwDestAddressSize;
    DWORD dwDestAddressOffset;
    // TAPI version 3.1
    DWORD dwCallerAddressType;
    DWORD dwDestAddressType
} LINEFORWARD, FAR *LPLINEFORWARD;
```

Parameters

```
dwForwardMode
```

The types of forwarding. This member uses one of the LINEFORWARDMODE_ constants.

```
dwCallerAddressSize
```

```
dwCallerAddressOffset
```

The size, in bytes, of the variably sized address field containing 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 is set to zero if dwForwardMode is not one of the following:

```
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 containing the address of the address where calls are to be forwarded, and the offset, in bytes, from the beginning of the containing data structure.

dwCallerAddressType

Windows XP: The address type of the caller. This member of the structure is available only if the negotiated version of TAPI is 3.1 or higher.

dwDestAddressType

Windows XP: The address type for the called destination. This member of the structure is available only if the negotiated version of TAPI is 3.1 or higher.

LINEMESSAGE

Description

The LINEMESSAGE structure contains parameter values specifying a change in status of the line the application currently has open. The lineGetMessage function returns the LINEMESSAGE structure.

Function Details

```
typedef struct linemessage_tag {
    DWORD    hDevice;
    DWORD    dwMessageID;
    DWORD_PTR dwCallbackInstance;
    DWORD_PTR dwParam1;
    DWORD_PTR dwParam2;
    DWORD_PTR dwParam3;
} LINEMESSAGE, FAR *LPLINEMESSAGE;
```

Parameters

hDevice

A handle to either a line device or a call. The nature of this handle (line handle or call handle) can be determined by the context provided by dwMessageID.

dwMessageID

A line or call device message.

dwCallbackInstance

Instance data passed back to the application, which was specified by the application in the dwCallBackInstance parameter of lineInitializeEx. This DWORD is not interpreted by TAPI.

dwParam1

A parameter for the message.

dwParam2

A parameter for the message.

dwParam3

A parameter for the message.



Cisco Unified CME TAPI Phone Device

Revised: January 12, 2007

The Cisco Unified TAPI implementation comprises a set of classes that expose the functionality of Cisco Unified IP Telephony Solutions. This API allows developers to create customized IP telephony applications for Cisco Unified CME without specific knowledge of the communication protocols between Cisco Unified CME and the TSP. For example, a developer could create a TAPI application that communicates with an external voice messaging system.

This chapter outlines the TAPI 2.1 functions, events, and messages that Cisco Unified TSP 2.1 supports. The Cisco Unified TAPI phone device implementation contains functions in the following areas:

- [Cisco Unified CME TAPI Phone Functions, page 88](#)
- [Cisco Unified CME TAPI Phone Messages, page 101](#)
- [Cisco Unified CME TAPI Phone Structures, page 104](#)

Cisco Unified CME TAPI Phone Functions

TAPI phone functions enable an application to control physical aspects of a phone.

phoneCallbackFunc

Description

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 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.

phoneClose

Description

The phoneClose function closes the specified open phone device.

Function Details

```
LONG phoneClose(  
  HPHONE hPhone  
);
```

Parameters

hPhone

A handle to the open phone device that is to be closed. If the function succeeds, the handle is no longer valid.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs. Possible return values are:

```
PHONEERR_INVALIDPHONEHANDLE,  
PHONEERR_OPERATIONFAILED.
```

phoneGetDevCaps

Description

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. This number is obtained 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 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.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs. Possible return values are:

```
PHONEERR_INVALIDHANDLE, PHONEERR_INVALIDPOINTER, PHONEERR_BADDEVICEID,
PHONEERR_INCOMPATIBLEAPIVERSION, PHONEERR_OPERATIONUNAVAIL, PHONEERR_NODEVICE.
```

phoneGetDisplay

Description

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.

The Display parameter is returned with the contents of the current prompt display line of the IP phone.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs. Possible return values are:

```
PHONEERR_INVALIDPHONEHANDLE, PHONEERR_OPERATIONFAILED,
```

phoneGetHookswitch

Description

The phoneGetHookSwitch function returns the current hookswitch mode of the specified open phone device.

Function Details

```
LONG phoneGetHookSwitch(  
  HPHONE hPhone,  
  LPDWORD lpdwHookSwitchDevs  
);
```

Parameters

hPhone

A handle to the open phone device.

`lpdwHookSwitchDevs`

Pointer to a DWORD to be filled with the mode of the phone's hookswitch devices. If a bit position is FALSE, the corresponding hookswitch device is onhook; if TRUE, the microphone and/or speaker part of the corresponding hookswitch device is offhook. To find out whether the microphone and/or speaker are enabled, the application can use `phoneGetStatus`.

Return Values

`PHONEERR_INVALIDPHONEHANDLE`, `PHONEERR_INVALIDPOINTER`, `PHONEERR_RESOURCEUNAVAIL`, `PHONEERR_INVALIDPHONESTATE`, `PHONEERR_OPERATIONUNAVAIL`, `PHONEERR_UNINITIALIZED`.

phoneGetMessage

Description

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 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 when 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

Possible return values follow:

`PHONEERR_INVALIDAPPHANDLE`, `PHONEERR_OPERATIONFAILED`, `PHONEERR_INVALIDPOINTER`, `PHONEERR_NOMEM`.

phoneGetRing

Description

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.

lpdwVolume

The volume level with which the phone is ringing.

phoneInitializeEx

Description

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 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 is 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 is 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 not 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 is supported by TAPI fills this location, thereby allowing the application to detect and adapt to having been 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

Possible return values follow:

```
PHONEERR_INVALIDAPPNAME, PHONEERR_OPERATIONFAILED,
PHONEERR_INIFILECORRUPT, PHONEERR_INVALIDPOINTER,
PHONEERR_REINIT, PHONEERR_NOMEM, PHONEERR_INVALIDPARAM.
```

phoneNegotiateAPIVersion

Description

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 is 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

Possible return values follow:

```
PHONEERR_INVALIDHANDLE, PHONEERR_OPERATIONFAILED,
PHONEERR_BADDEVICEID, PHONEERR_OPERATIONUNAVAIL,
PHONEERR_NODRIVER, PHONEERR_NOMEM,
```

```
PHONEERR_INVALIDPOINTER,
PHONEERR_RESOURCEUNAVAIL,
PHONEERR_INCOMPATIBLEAPIVERS
```

phoneOpen

Description

The `phoneOpen` function opens the specified phone device. The device can be opened 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.

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 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 is informed about events and state changes occurring 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.

phoneSetHookswitch

Description

The `phoneSetHookSwitch` function sets the hook state of the specified open phone's hookswitch devices to the specified mode. Only the hookswitch state of the hookswitch devices listed is affected.

Function Details

```
LONG WINAPI phoneSetHookSwitch(
    HPHONE hPhone,
    DWORD dwHookSwitchDevs,
    DWORD dwHookSwitchMode
);
```

Parameters

`hPhone`

Handle to the open phone device. The application must be the owner of the phone.

`dwHookSwitchDevs`

Device whose hookswitch mode is to be set. This parameter uses one and only one of the `PHONEHOOKSWITCHDEV_` constants:

- `PHONEHOOKSWITCHDEV_HANDSET`: the phone's handset.
- `PHONEHOOKSWITCHDEV_SPEAKER`: the phone's speakerphone or adjunct.
- `PHONEHOOKSWITCHDEV_HEADSET`: the phone's headset.

`dwHookSwitchMode`

Hookswitch mode to set. This parameter uses one and only one of the `PHONEHOOKSWITCHMODE_` constants:

- `PHONEHOOKSWITCHMODE_ONHOOK`: The device's microphone and speaker are both onhook.
- `PHONEHOOKSWITCHMODE_MIC`: The device's microphone is active, the speaker is mute.

- **PHONEHOOKSWITCHMODE_SPEAKER:** The device's speaker is active, the microphone is mute.
- **PHONEHOOKSWITCHMODE_MICSPEAKER:** The device's microphone and speaker are both active.

Return Values

Possible return values are:

```
PHONEERR_INVALIDPHONEHANDLE, PHONEERR_OPERATIONUNAVAIL, PHONEERR_NOTOWNER,
PHONEERR_NOMEM, PHONEERR_INVALIDHOOKSWITCHDEV, PHONEERR_RESOURCEUNAVAIL,
PHONEERR_INVALIDHOOKSWITCHMODE, PHONEERR_OPERATIONFAILED, PHONEERR_INVALIDPHONESTATE,
PHONEERR_UNINITIALIZED.
```

phoneSetRing

Description

The `phoneSetRing` function rings the specified open phone device using the specified ring mode and volume.



Note

For Cisco Unified CME TSP 2.1, this function is used only to set the ringer volume. The Ringer Device is selected during configuration and the ring patterns are played based on the ring type sent by the Cisco Unified CME.

Function Details

```
LONG WINAPI phoneSetRing(
    HPHONE hPhone,
    DWORD dwRingMode,
    DWORD dwVolume
);
```

Parameters

`hPhone`

Handle to the open phone device. The application must be the owner of the phone device.

`dwRingMode`

Ringing pattern with which to ring the phone. This parameter must be within the range of zero to the value of the `dwNumRingModes` member in the `PHONECAPS` structure. If `dwNumRingModes` is zero, the ring mode of the phone cannot be controlled; if `dwNumRingModes` is 1, a value of 0 for `dwRingMode` indicates that the phone should not be rung (silence), and other values from 1 to `dwNumRingModes` are valid ring modes for the phone device.

`dwVolume`

Volume level with which the phone is ringing. This is a number in the range 0x00000000 (silence) to 0x0000FFFF (maximum volume). The actual granularity and quantization of volume settings in this range are service provider-specific. A value for dwVolume that is out of range is set to the nearest value in the range.

Return Values

```
PHONEERR_INVALIDPHONEHANDLE, PHONEERR_NOMEM, PHONEERR_NOTOWNER,
PHONEERR_RESOURCEUNAVAIL, PHONEERR_INVALIDPHONESTATE, PHONEERR_OPERATIONFAILED,
PHONEERR_INVALIDRINGMODE, PHONEERR_UNINITIALIZED, PHONEERR_OPERATIONUNAVAIL.
```

phoneSetVolume

Description

The phoneSetVolume function sets the volume of the speaker component of the specified hookswitch device to the specified level.

Function Details

```
LONG WINAPI phoneSetVolume(
    HPHONE hPhone,
    DWORD dwHookSwitchDev,
    DWORD dwVolume
);
```

Parameters

hPhone

Handle to the open phone device. The application must be the owner of the phone.

dwHookSwitchDev

Hookswitch device whose speaker's volume is to be set, one of the PHONEHOOKSWITCHDEV_ constants.

dwVolume

New volume setting of the device. The dwVolume parameter specifies the volume level of the hookswitch device. This is a number in the range 0x00000000 (silence) to 0x0000FFFF (maximum volume). The actual granularity and quantization of volume settings in this range are service provider-specific. A value for dwVolume that is out of range is set to the nearest value in the range.

Return Values

```
PHONEERR_INVALIDPHONEHANDLE, PHONEERR_NOMEM, PHONEERR_NOTOWNER,
PHONEERR_RESOURCEUNAVAIL, PHONEERR_INVALIDPHONESTATE, PHONEERR_OPERATIONFAILED,
PHONEERR_INVALIDHOOKSWITCHDEV, PHONEERR_UNINITIALIZED, PHONEERR_OPERATIONUNAVAIL.
```

phoneShutdown

Description

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  
);
```

Parameters

hPhoneApp

The application usage handle for TAPI.

Return Values

Returns zero if the request succeeds or a negative error number if an error occurs.

Possible return values follow:

```
PHONEERR_INVALIDAPPHANDLE, PHONEERR_NOMEM,  
PHONEERR_UNINITIALIZED,
```

Cisco Unified CME TAPI Phone Messages

This section describes the phone device messages that Cisco Unified CME TSP 2.1 supports. These messages notify the application of asynchronous events such as the a new call arriving in the Cisco Unified CME. The messages get sent to the application using the method that the application specifies in `phoneInitializeEx`.

PHONE_CLOSE

Description

The TAPI `PHONE_CLOSE` message is sent when an open phone device has been forcibly closed as part of resource reclamation. The device handle is no longer valid once this message has been 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 has been sent.

`dwCallbackInstance`

The application's callback instance that provided when opening the phone device.

`dwParam1`

Unused.

`dwParam2`

Unused.

`dwParam3`

Unused.

PHONE_REMOVE

Description

The TAPI PHONE_REMOVE message is sent to inform an application of the removal (deletion from the system) of a phone device. Generally, this is not used for temporary removals, such as extraction of PCMCIA devices, but only for permanent removals in which the device would no longer be reported by the service provider if TAPI were re-initialized.

Function Details

```
PHONE_REMOVE
hDevice = (DWORD) 0;
dwCallbackInstance = (DWORD) 0;
dwParam1 = (DWORD) hDeviceID;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

hDevice

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

Description

The TAPI PHONE_REPLY message is 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

Unused.

dwCallbackInstance

Returns the application's 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

Unused.

Cisco Unified CME TAPI Phone Structures

This section describes the main phone structures that are impacted by Cisco Unified CME TSP 2.1. These structures are used to exchange parameters between the application and TAPI and between TAPI and Cisco Unified CME TSP 2.1. In response to the line messages from the TSP, the TAPI layer makes functions calls with these structures to obtain the message-related detailed information.

PHONECAPS

Description

The PHONECAPS structure describes the capabilities of a phone device. The phoneGetDevCaps and TSPI_phoneGetDevCaps functions return this structure.

Function Details

```
typedef struct phonecaps_tag {
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    DWORD dwProviderInfoSize;
    DWORD dwProviderInfoOffset;
    DWORD dwPhoneInfoSize;
    DWORD dwPhoneInfoOffset;
    DWORD dwPermanentPhoneID;
    DWORD dwPhoneNameSize;
    DWORD dwPhoneNameOffset;
    DWORD dwStringFormat;
    DWORD dwPhoneStates;
    DWORD dwHookSwitchDevs;
    DWORD dwHandsetHookSwitchModes;
    DWORD dwSpeakerHookSwitchModes;
    DWORD dwHeadsetHookSwitchModes;
    DWORD dwVolumeFlags;
    DWORD dwGainFlags;
    DWORD dwDisplayNumRows;
    DWORD dwDisplayNumColumns;
    DWORD dwNumRingModes;
    DWORD dwNumButtonLamps;
    DWORD dwButtonModesSize;
    DWORD dwButtonModesOffset;
    DWORD dwButtonFunctionsSize;
    DWORD dwButtonFunctionsOffset;
    DWORD dwLampModesSize;
    DWORD dwLampModesOffset;
    DWORD dwNumSetData;
    DWORD dwSetDataSize;
    DWORD dwSetDataOffset;
    DWORD dwNumGetData;
    DWORD dwGetDataSize;
    DWORD dwGetDataOffset;
    DWORD dwDevSpecificSize;
    DWORD dwDevSpecificOffset;
    DWORD dwDeviceClassesSize;
    DWORD dwDeviceClassesOffset;
    DWORD dwPhoneFeatures;
    DWORD dwSettableHandsetHookSwitchModes;
}
```

```

    DWORD dwSettableSpeakerHookSwitchModes;
    DWORD dwSettableHeadsetHookSwitchModes;
    DWORD dwMonitoredHandsetHookSwitchModes;
    DWORD dwMonitoredSpeakerHookSwitchModes;
    DWORD dwMonitoredHeadsetHookSwitchModes;
    GUID PermanentPhoneGuid;
} PHONECAPS,

```

Parameters

`dwTotalSize`

Total size allocated to this data structure, in bytes.

`dwNeededSize`

Size for this data structure that is needed to hold all the returned information, in bytes.

`dwUsedSize`

Size of the portion of this data structure that contains useful information, in bytes.

`dwProviderInfoSize`

Size of the provider-specific information, in bytes. If the provider-specific information is a pointer to a string, the size must include the null terminator.

`dwProviderInfoOffset`

Offset from the beginning of the structure to the variably sized field containing service provider-specific information.

This member provides information about the provider hardware and/or software, such as the vendor name and version numbers of hardware and software. This information can be useful when a user needs to call customer service with problems regarding the provider. The size of the field is specified by `dwProviderInfoSize`.

`dwPhoneInfoSize`

Size of the phone-specific information, in bytes. If the phone-specific information is a pointer to a string, the size must include the null terminator.

`dwPhoneInfoOffset`

Offset from the beginning of the structure to the variably sized device field containing phone-specific information.

This member provides information about the attached phone device, such as the phone device manufacturer, the model name, the software version, and so on. This information can be useful when a user needs to call customer service with problems regarding the phone. The size of the field is specified by `dwPhoneInfoSize`.

`dwPermanentPhoneID`

Permanent identifier by which the phone device is known in the system's configuration.

`dwPhoneNameSize`

Size of the name for the phone, including the null terminator, in bytes.

`dwPhoneNameOffset`

Offset from the beginning of the structure to the variably sized device field containing a user-specified name for this phone device. This name can be configured by the user when configuring the phone device's service provider and is provided for the user's convenience. The size of the field is specified by `dwPhoneNameSize`.

`dwStringFormat`

String format to be used with this phone device. This member uses one of the `STRINGFORMAT_` constants.

`dwPhoneStates`

State changes for this phone device for which the application can be notified in a `PHONE_STATE` message. This member one or more of the `PHONESTATE_` constants.

`dwHookSwitchDevs`

Phone's hookswitch devices. This member uses one of the `PHONEHOOKSWITCHDEV_` constants.

`dwHandsetHookSwitchModes`

Hookswitch mode of the handset. The member is only meaningful if the hookswitch device is listed in `dwHookSwitchDevs`. It uses one of the `PHONEHOOKSWITCHMODE_` constants.

`dwSpeakerHookSwitchModes`

Hookswitch mode of the speaker. The member is only meaningful if the hookswitch device is listed in `dwHookSwitchDevs`. It uses one of the `PHONEHOOKSWITCHMODE_` constants.

`dwHeadsetHookSwitchModes`

Hookswitch mode of the headset. The member is only meaningful if the hookswitch device is listed in `dwHookSwitchDevs`. It uses one of the `PHONEHOOKSWITCHMODE_` constants.

`dwVolumeFlags`

Volume-setting capabilities of the phone device's speaker components. If the bit in position `PHONEHOOKSWITCHDEV_` is `TRUE`, the volume of the corresponding hookswitch device's speaker component can be adjusted with `phoneSetVolume`.

`dwGainFlags`

Gain-setting capabilities of the phone device's microphone components. If the bit position `PHONEHOOKSWITCHDEV_` is `TRUE`, the volume of the corresponding hookswitch device's microphone component can be adjusted with `phoneSetGain`.

`dwDisplayNumRows`

Display capabilities of the phone device by describing the number of rows in the phone display. The `dwDisplayNumRows` and `dwDisplayNumColumns` members are both zero for a phone device without a display.

`dwDisplayNumColumns`

Display capabilities of the phone device by describing the number of columns in the phone display. The `dwDisplayNumRows` and `dwDisplayNumColumns` members are both zero for a phone device without a display.

`dwNumRingModes`

Ring capabilities of the phone device. The phone is able to ring with `dwNumRingModes` different ring patterns, identified as 1, 2, through `dwNumRingModes` minus one. If the value of this member is 0, applications have no control over the ring mode of the phone. If the value of this member is greater than

0, it indicates the number of ring modes in addition to silence that are supported by the service provider. A value of 0 in the `lpdwRingMode` parameter of `phoneGetRing` or the `dwRingMode` parameter of `phoneSetRing` indicates silence (the phone is not ringing or should not be rung), and `dwRingMode` values of 1 to `dwNumRingModes` are valid ring modes for the phone device.

`dwNumButtonLamps`

Number of button/lamps on the phone device that are detectable in TAPI. Button/lamps are identified by their identifier. Valid button/lamp identifiers range from zero to `dwNumButtonLamps` minus one. The keypad buttons '0', through '9', '*', and '#' are assigned the identifiers 0 through 12.

`dwButtonModesSize`

Size of the button modes array, in bytes.

`dwButtonModesOffset`

Offset from the beginning of this structure to the variably sized field containing the button modes of the phone's buttons. The array is indexed by button/lamp identifier. This array uses the `PHONEBUTTONMODE_` constants. The size of the array is specified by `dwButtonModesSize`.

`dwButtonFunctionsSize`

Size of the button functions field, in bytes.

`dwButtonFunctionsOffset`

Offset from the beginning of this structure to the variably sized field containing the button functions of the phone's buttons. The array is indexed by button/lamp identifier. This array uses the `PHONEBUTTONFUNCTION_` constants. The size of the array is specified by `dwButtonFunctionsSize`.

`dwLampModesSize`

Size of the lamp modes array, in bytes.

`dwLampModesOffset`

Offset from the beginning of this structure to the variably sized field containing the lamp modes of the phone's lamps. The array is indexed by button/lamp identifier. This array uses the `PHONELAMPMODE_` constants. The size of the array is specified by `dwLampModesSize`.

`dwNumSetData`

Number of different download areas in the phone device. The different areas are referred to using the data IDs 0, 1, and `dwNumSetData` minus one. If this member is zero, the phone does not support the download capability.

`dwSetDataSize`

Size of the data size array, in bytes.

`dwSetDataOffset`

Offset from the beginning of this structure to the variably sized field containing the sizes (in bytes) of the phone's download data areas. This is an array with `DWORD`-sized elements indexed by data identifier. The size of the array is specified by `dwSetDataSize`.

`dwNumGetData`

Number of different upload areas in the phone device. The different areas are referred to using the data IDs 0, 1, and `dwNumGetData` minus one. If this field is zero, the phone does not support the upload capability.

`dwGetDataSize`

Size of the data size array, in bytes.

`dwGetDataOffset`

Offset from the beginning of this structure to the variably sized field containing the sizes (in bytes) of the phone's upload data areas. This is an array with DWORD-sized elements indexed by data identifier. The size of the array is specified by `dwGetDataSize`.

`dwDevSpecificSize`

Size of the device-specific field, in bytes. If the device specific information is a pointer to a string, the size must include the null terminator.

`dwDevSpecificOffset`

Offset from the beginning of this structure to the variably sized device-specific field. The size of the field is specified by `dwDevSpecificSize`.

`dwDeviceClassesSize`

Size of the supported device class identifiers, in bytes.

`dwDeviceClassesOffset`

Offset from the beginning of this structure to a string consisting of the device class identifiers supported on this device for use with `phoneGetID`. The identifiers are separated by NULLs, and the last identifier in the list is followed by two NULLs. The size of the field is specified by `dwDeviceClassesSize`.

`dwPhoneFeatures`

Flags that indicate which telephony API functions can be invoked on the phone. A zero indicates the corresponding feature is not implemented and can never be invoked by the application on the phone; a one indicates the feature may be invoked depending on the device state and other factors. This member uses `PHONEFEATURE_` constants.

`dwSettableHandsetHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be set on the handset using `phoneSetHookSwitch`.

`dwSettableSpeakerHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be set on the speakerphone using `phoneSetHookSwitch`.

`dwSettableHeadsetHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be set on the headset using `phoneSetHookSwitch`.

`dwMonitoredHandsetHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be detected and reported for the handset in a `PHONE_STATE` message and by `phoneGetHookSwitch`.

`dwMonitoredSpeakerHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be detected and reported for the speakerphone in a `PHONE_STATE` message and by `phoneSetHookSwitch`.

`dwMonitoredHeadsetHookSwitchModes`

`PHONEHOOKSWITCHMODE_` values that can be detected and reported for the headset in a `PHONE_STATE` message and by `phoneSetHookSwitch`.

`PermanentPhoneGuid`

The GUID permanently associated with this phone.

PHONEEXTENSIONID

Description

The PHONEEXTENSIONID structure describes an extension identifier. Extension identifiers are used to identify service provider-specific extensions for phone device classes. The phoneNegotiateAPIVersion and TSPI_phoneGetExtensionID functions return this structure.

Function Details

```
typedef struct phoneextensionid_tag {
    DWORD dwExtensionID0;
    DWORD dwExtensionID1;
    DWORD dwExtensionID2;
    DWORD dwExtensionID3;
} PHONEEXTENSIONID,
*LPPHONEEXTENSIONID;
```

Parameters

dwExtensionID0

First part of the extension identifier.

dwExtensionID1

Second part of the extension identifier.

dwExtensionID2

Third part of the extension identifier.

dwExtensionID3

Fourth part of the extension identifier.

PHONEMESSAGE

Description

The PHONEMESSAGE structure contains the next message queued for delivery to the application. The phoneGetMessage function returns this structure.

Function Details

```
typedef struct phonemessage_tag {
    DWORD hDevice;
    DWORD dwMessageID;
    DWORD_PTR dwCallbackInstance;
    DWORD_PTR dwParam1;
```

```

        DWORD_PTR dwParam2;
        DWORD_PTR dwParam3;
    } PHONEMESSAGE,
    *LPPHONEMESSAGE;

```

Parameters

hDevice

Handle to a phone device.

dwMessageID

Phone message.

dwCallbackInstance

Instance data passed back to the application, which was specified by the application in phoneInitializeEx. This value is not interpreted by TAPI.

dwParam1

Parameter for the message.

dwParam2

Parameter for the message.

dwParam3

Parameter for the message.



A

Application Programming Interface (API) [2](#)
automatic call distribution (ACD) [1](#)

C

call handle [15](#)
Cisco [1](#)
Cisco Unified CME endpoints [5](#)
Cisco Unified CME TSP [1](#)
Cisco Unified CME TSP-based solution [1](#)
Cisco Unified IP Telephony Solutions [1](#)
connects [9](#)
Consultation-Transfer [13](#)
customized IP Telephony [11](#)

D

debug tracing [9](#)
Device ID [10](#)
device types [8](#)
Dual Line DNs [8](#)
DWORD type [15](#)

F

First-Party Call Control [6](#)

H

HANDLE type [15](#)

I

ISDN Q.931 [69](#)

L

LINE_ADDRESSSTATE [44](#)
LINE_APPNEWCALL [45](#)
LINE_CALLINFO [46, 63](#)
LINE_CALLSTATE [47](#)
LINE_CLOSE [49](#)
LINE_CLOSE message [9](#)
LINE_CREATE [50](#)
LINE_DEVSPECIFIC [51](#)
LINE_LINEDEVSTATE [52](#)
LINE_MONITORMEDIA [66](#)
LINE_REINIT message [9](#)
LINE_REMOVE [53](#)
LINE_REPLY [53](#)
LINEADDRCAPFLAGS_ [58](#)
LINEADDRESSCAPS [55](#)
LINEADDRESSMODE_ [79](#)
LINEADDRESSMODE_ADDRESSID [73](#)
LINEADDRESSSHARING_ [56](#)
LINEADDRESSSTATE [23](#)
LINEADDRESSSTATE_CAPSCHANGE [45](#)
LINEADDRESSSTATE_DEVSPECIFIC [45](#)
LINEADDRESSSTATE_FORWARD [45](#)
LINEADDRESSSTATE_INUSEMANY [45](#)
LINEADDRESSSTATE_INUSEONE [45](#)
LINEADDRESSSTATE_INUSEZERO [45](#)
LINEADDRESSSTATE_NUMCALLS [45](#)
LINEADDRESSSTATE_OTHER [45](#)

- LINEADDRESSSTATE_TERMINALS 45
- LINEADDRESSSTATUS 61
- LINEADDRFEATURE_ 59
- lineAddtoConference 11
- lineAnswer 12
- LINEAPPINFO 62
- LINEBEARERMODE_ 72
- LINEBEARERMODE_VOICE 72
- lineBlindTransfer 13
- LINEBUSYMODE_ 57
- lineCallbackFunc 14
- LINECALLCOMPLCOND_ 59
- LINECALLCOMPLMODE_ 59
- LINECALLCOMPLMODE_MESSAGE 59
- LINECALLFEATURE_ 58
- LINECALLFEATURE2_ 60
- LINECALLHUBTRACKING 82
- LINECALLINFO 63
- LINECALLINFOSTATE_ 46
- LINECALLLIST 70
- LINECALLORIGIN_ 66
- LINECALLPARAMFLAGS_ 66
- LINECALLPARAMS 71
- LINECALLPARTYID_ 57, 67
- LINECALLPRIVILEGE 36
- LINECALLPRIVILEGE_ 49
- LINECALLPRIVILEGE_MONITOR 36
- LINECALLPRIVILEGE_NONE 36
- LINECALLPRIVILEGE_OWNER 36
- LINECALLREASON_ 66
- LINECALLREASON_CALLCOMPLETION 66
- LINECALLSELECT_ 30
- LINECALLSELECT_ADDRESS 30
- LINECALLSELECT_CALL 30
- LINECALLSELECT_LINE 30
- LINECALLSTATE_ 57
- LINECALLSTATE_ACCEPTED 47
- LINECALLSTATE_CONFERENCED 47
- LINECALLSTATE_CONNECTED 48
- LINECALLSTATE_DIALING 47
- LINECALLSTATE_DIALTONE 47
- LINECALLSTATE_IDLE 47
- LINECALLSTATE_OFFERING 47
- LINECALLSTATE_ONHOLD 48
- LINECALLSTATE_ONHOLDPENDCONF 47
- LINECALLSTATE_ONHOLDPENDTRANSFER 48
- LINECALLSTATE_PROCEEDING 48
- LINECALLSTATE_RINGBACK 47
- LINECALLSTATE_UNKNOWN 48
- LINECALLSTATUS 76
- LINECALLTREATMENTENTRY 60
- lineClose 15
- lineCompleteCall 16
- lineCompleteTransfer 17
- lineConfigProvider 19
- LINECONNECTEDMODE_ACTIVE 48
- LINECONNECTEDMODE_INACTIVE 48
- LINEDEVCAPFLAGS_ 80
- LINEDEVCAPS 77
- lineDevSpecificFeature 20
- LINEDEVSTATE_ 52
- LINEDEVSTATE_OUTOFSERVICE 48
- LINEDEVSTATE_REINIT 52
- LINEDEVSTATE_RINGING 52
- LINEDEVSTATE_TRANSLATECHANGE 52
- LINEDEVSTATUS 83
- LINEDEVSTATUSFLAGS_ 84
- lineDial 21
- LINEDIALPARAMS. 66
- LINEDIALTONEMODE_UNAVAIL 48
- LINEDISCONNECTMODE_ 49
- LINEDISCONNECTMODE_BUSY 49
- LINEDISCONNECTMODE_CONGESTION 49
- LINEDISCONNECTMODE_FACCMC 49
- LINEDISCONNECTMODE_NOANSWER 49
- LINEDISCONNECTMODE_NORMAL 49
- LINEDISCONNECTMODE_REJECT 49
- LINEDISCONNECTMODE_UNAVAIL 49

LINEDISCONNECTMODE_UNKNOWN 49
 lineDrop 22
 LINEERR_INVALCALLHANDLE 12
 LINEEXTENSIONID 35
 LINEFEATURE_ 82
 LINEFORWARD 85
 lineForward 23
 LINEFORWARDLIST 85
 LINEFORWARDMODE_ 85
 LINEGENERATETONE 79
 lineGetAddressCaps 25
 lineGetAddressStatus 27
 lineGetCallInfo 28
 lineGetCallStatus 28
 lineGetDevCaps 29
 lineGetID 30
 line handle 15
 lineHold 31
 lineInitializeEx 32
 LINEINITIALIZEEXPARAMS 33
 lineMakeCall 33
 LINEMEDIAMODE_ 66
 LINEMEDIAMODE_INTERACTIVEVOICE 73
 LINEMESSAGE 86
 LINEMONITORTONE 80
 lineNegotiateAPIVersion 34
 LINEOFFERINGMODE_ACTIVE 49
 lineOpen 35
 LINEOPENOPTION_SINGLEADDRESS 63
 linePark 37
 LINEPARKMODE_ 58
 LINEPARKMODE_Constants 37
 linePickup 38
 LINEREMOVEFROMCONF_ 58
 lineRemoveProvider 39
 LINEREQUESTMODE_ 54
 LINEREQUESTMODE_DROP 54
 LINEROAMMODE_ 84
 lineSetupConference 40

lineSetupTransfer 41
 LINESPECIALINFO_ 57
 LINETERMCAPS 81
 LINETERMMODE_ 70
 LINETRANSFERMODE_ 18
 LINETRANSFERMODE_CONFERENCE 18
 LINETRANSFERMODE_TRANSFER 18
 LINETRANSLATEOUTPUT 69
 lineUnhold 42
 lineUnpark 42

M

MAC Address 10
 Monitored DNs 8

P

PHONE_CLOSE 101
 PHONE_REMOVE 102
 PHONE_REPLY 103
 PHONE_STATE 106
 PHONEBUTTONFUNCTION_ 20
 phoneCallbackFunc 88
 PHONECAPS 104
 phoneClose 89
 PHONEEXTENSIONID 95, 109
 PHONEFEATURE_ 108
 phoneGetDevCaps 90, 104
 phoneGetDisplay 91
 phoneGetHookSwitch 91
 phoneGetMessage 92
 phoneGetRing 93
 PHONEHOOKSWITCHDEV_ 97
 PHONEHOOKSWITCHDEV_HANDSET 97
 PHONEHOOKSWITCHDEV_HEADSET 97
 PHONEHOOKSWITCHDEV_SPEAKER 97
 PHONEHOOKSWITCHMODE_ 97

PHONEHOOKSWITCHMODE_MIC 97
 PHONEHOOKSWITCHMODE_MICSPKAKER 98
 PHONEHOOKSWITCHMODE_ONHOOK 97
 PHONEHOOKSWITCHMODE_SPEAKER 98
 phoneInitializeEx 93
 PHONEINITIALIZEEXOPTION_USEEVENT 92
 PHONEINITIALIZEEXPARAMS 92
 PHONEMESSAGE 92, 109
 phoneNegotiateAPIVersion 95, 109
 phoneOpen 96
 PHONEPRIVILEGE_ 97
 PHONEPRIVILEGE_MONITOR 97
 PHONEPRIVILEGE_OWNER 97
 phoneSetHookSwitch 97
 phoneSetRing 98
 phoneSetVolume 99
 phoneShutdown 100
 PHONESTATE_ 106

R

Required Software 3

S

Shared Lines 8
 Single Line DNs 8
 Startup with Windows 9
 STRINGFORMAT_ 79
 STRINGFORMAT_ASCII 37
 Supported Device and Line Types 8

T

TAPI 2.2 service provider 8
 TAPI Line Functions 11
 TAPISRV 7
 Third-Party Call Control 6

TSPI_phoneGetDevCaps 104
 TSPI_phoneGetExtensionID 109
 TSPI_providerConfig 19

V

VARSTRING 31, 37

W

WinSock2 70