TAPI Developer Guide for Cisco CME/SRST

Revised 09/1/2005

General Information

Introduction

Telephony Application Programmer's Interface (TAPI) is the set of classes and principles of operation that constitute a telephony application programming interface. TAPI implementations are the interface between computer telephony applications and telephony services.

The Cisco TAPI Service Provider (TSP) allows PC software developers to create customized IP telephony applications for Cisco IP Phone users; for example, Contact Management applications that require telephony integration to handle inbound and outbound telephone calls. The Cisco TSP enables the Cisco IP Telephony system to integrate with Microsoft Windows™ user-level applications that support the Microsoft Windows TAPI standard.

The Cisco TAPI implementation uses the Microsoft™ TAPI specification to support Cisco IP Telephony Solutions. To enable a Cisco TAPI-based solutions, the following are essential:

- TAPI support/service running on the operating system
- A TAPI-based software application
- A Cisco IP Telephony phone system

Purpose

This document describes the Cisco TAPI implementation for Cisco CallManager Express (CME) and Cisco Survivable Site Telephony (SRST), detailing the functions that comprise the implementation software and illustrating how to use these functions to create applications that support the Cisco IP Telephony hardware, software, and processes.

One of the primary goals 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 TAPI implementation is designed to conform as closely as possible to the TAPI specification.

Audience

This document is intended for telephony software engineers who are developing Cisco telephony applications that require TAPI. This document assumes that the engineer is familiar with both the C or C++ languages and the Microsoft™ TAPI specification.

Related Documentation

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

- Microsoft TAPI 2.1 Features:

  http://www.microsoft.com/ntserver/techresources/commnet/tele/tapi21.asp
- Getting Started with Windows Telephony

  http://www.microsoft.com/ntserver/techresources/commnet/tele/getstartedtele.asp
- Creating Next Generation Telephony Applications:

  http://www.microsoft.com/ntserver/techresources/commnet/tele/tapi21wp.asp
- The Microsoft Telephony Application Programming Interface (TAPI) Programmer's Reference

Conventions

This document uses the following conventions:

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in *italic*s.
[ ]	Elements in square brackets are optional.
{ x | y | z }	Alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	An unquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`screen` font	Terminal sessions and information the system displays are in `screen` font.
boldface screen font	Information you must enter is in **boldface screen** font.
italic screen	Arguments for which you supply values are in *italic screen* font.

Cisco Connection Online

Cisco Connection Online (CCO) is Cisco Systems' primary, real-time support channel. Maintenance customers and partners can self-register on CCO to obtain additional information and services.

Available 24 hours a day, 7 days a week, CCO provides a wealth of standard and value-added services to Cisco's customers and business partners. CCO services include product information, product documentation, software updates, release notes, technical tips, the Bug Navigator, configuration notes, brochures, descriptions of service offerings, and download access to public and authorized files.

CCO serves a wide variety of users through two interfaces that are updated and enhanced simultaneously: a character-based version and a multimedia version that resides on the World Wide Web (WWW). The character-based CCO supports Zmodem, Kermit, Xmodem, FTP, and Internet e-mail, and it is excellent for quick access to information over lower bandwidths. The WWW version of CCO provides richly formatted documents with photographs, figures, graphics, and video, as well as hyperlinks to related information.

You can access CCO in the following ways:

- WWW: http://www.cisco.com
- WWW: http://www-europe.cisco.com
- WWW: http://www-china.cisco.com
- Telnet: cco.cisco.com
- Modem: From North America, 408 526-8070; from Europe, 33 1 64 46 40 82. Use the following terminal settings: VT100 emulation; databits: 8; parity: none; stop bits: 1; and connection rates up to 28.8 kbps.

For a copy of CCO's Frequently Asked Questions (FAQ), contact cco-help@cisco.com. For additional information, contact cco-team@cisco.com.

Note If you are a network administrator and need personal technical assistance with a Cisco product that is under warranty or covered by a maintenance contract, contact Cisco's Technical Assistance Center (TAC) at 800 553-2447, 408 526-7209, or tac@cisco.com. To obtain general information about Cisco Systems, Cisco products, or upgrades, contact 800 553-6387, 408 526-7208, or cs-rep@cisco.com.

Documentation CD-ROM

Cisco documentation and additional literature are available in a CD-ROM package, which ships with your product. The Documentation CD-ROM, a member of the Cisco Connection Family, is updated monthly. Therefore, it might be more current than printed documentation. To order additional copies of the Documentation CD-ROM, contact your local sales representative or call customer service. The CD-ROM package is available as a single package or as an annual subscription. You can also access Cisco documentation on the World Wide Web at http://www.cisco.com, http://www-china.cisco.com, or http://www-europe.cisco.com.

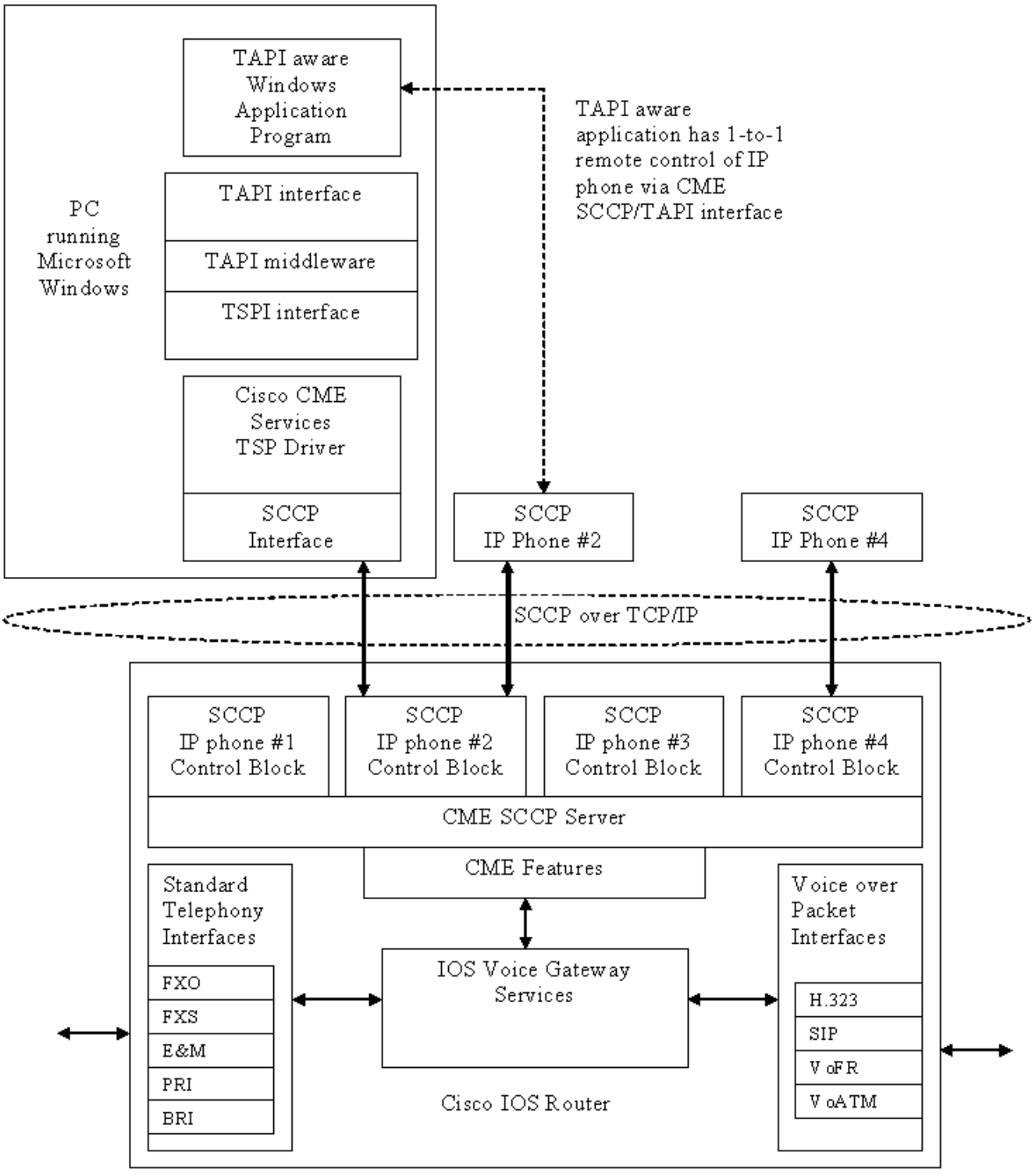
If you are reading Cisco product documentation on the World Wide Web, you can submit comments electronically. Click Feedback in the toolbar and select Documentation. After you complete the form, click Submit to send it to Cisco. We appreciate your comments.

Architecture

Cisco CallManager Express provides support for SCCP-based IP phones, allowing access to IOS Voice Gateway services, to IOS telephony interfaces for connection to the PSTN, and to Voice-over-Packet interfaces for VoIP calls using H.323 and SIP. See Figure 1.

From the perspective of most, general, IOS Voice Gateway services, the SCCP-based IP phones appear as normal FXS-port-connected analog phones. This enables IOS Voice Gateway services to route calls to and from SCCP IP phones in the same way it routes calls to and from analog phones that are directly connected to FXS voice ports on the router. This also allows SCCP IP phones to benefit from and leverage the standard IOS Voice Gateway VoIP protocols for making VoIP calls across the WAN/Internet.

Figure 1 CME TAPI Architecture



The Cisco CME TAPI interface allows a TAPI-aware application program running under Microsoft Windows™ on a Personal Computer to assert simple 1-to-1 remote control of an individual IP phone.

Note The TAPI interface does not allow a single PC to simultaneously control multiple IP phones.

The TAPI interface is designed to support only signaling operations, such as placing and receiving calls. The TAPI interface does not support the sending of voice media packets to and from the PC.

The TAPI interface can be used to control the on-hook and off-hook state of a single phone line on a single IP phone. It also supports the sending of dialing digits from the PC to Cisco CME so that the PC can instruct the Cisco CME to place an outbound call using the IP phone, as in address-book dialing.

When an incoming call is presented to the SCCP phone, a notification message can also be sent to the PC that includes incoming caller-id information. The TAPI-aware application can use this information to cause a "screen-pop" notification of the incoming call to PC user. The PC can then instruct the IP phone to answer the call (in speakerphone mode or with a headset), or reject the call and forward it elsewhere, such as to voicemail.

The Cisco CME TAPI interface is designed to support personal productivity software applications running on an individual PC, and to help automate the handling of incoming and outgoing calls to an IP phone.

The Microsoft Windows™ TSP driver for Cisco CME plugs into the TSPI layer interface provided by Microsoft TAPI and converts the TSPI function calls into SCCP control messages that are sent across TCP/IP to the IOS router. SCCP messages indicate operations such as on-hook, off-hook, keypad-digit press, and soft-key press. The Cisco CME router also sends SCCP messages to the TSP driver to indicate operations such as ringer-on, ringer-off, dial-tone, call-display information, and call state.

Restrictions

The following restrictions apply:

- The TSP does not support the sending of media from the router to the PC.
- The TAPI interface does not allow a single PC to simultaneously control multiple IP phones.

TAPI/TSPI Implementation

Table 1 lists the supported TAPI/TSPI functions. For more details about these functions, see the Microsoft™ TSPI reference.

Table 1 Supported TAPI/TSPI Line Functions at a Glance

TAPI Function	TSPI Function	Description
lineAnswer	TSPI_lineAnswer	Answers the specified offering call.
lineAddToConference	TSPI_lineAddToConference	
lineBlindTransfer	TSPI_lineBlindTransfer	Performs a blind or single-step transfer of the specified call to the specified destination address.
lineClose	TSPI_lineCloseCall	Closes the specified open line device after completing or aborting all outstanding calls and asynchronous operations on the device.
lineCompleteTransfer	TSPI_lineCompleteTransfer	Completes the transfer of the specified call to the party connected in the consultation call.

Table 1 Supported TAPI/TSPI Line Functions at a Glance (continued)

TAPI Function	TSPI Function	Description
lineDial	TSPI_lineDial	Dials the specified dialable number on the specified call.
lineDrop	TSPI_lineDrop	Drops or disconnects the specified call.
lineGetAddressID	TSPI_lineGetAddressID	Returns the address identifier associated with address in a different format on the specified line.
	TSPI_lineGetCallAddressID	Retrieves the address identifier for the indicated call.
lineGetCallInfo	TSPI_lineGetCallInfo	Returns detailed information about the specified call.
lineGetCallStatus	TSPI_lineGetCallStatus	Returns the current status of the specified call.
lineGetDevConfig	TSPI_lineGetDevConfig	Returns a data structure object, the contents of which are specific to the line (service provider) and device class, giving the current configuration of a device associated one-to-one with the line device.
	TSPI_lineGetExtensionID	Returns the extension identifier that the service provider supports for the indicated line device.
lineGetID	TSPI_lineGetID	Returns a device identifier for the specified device class associated with the selected line, address, or call.
	TSPI_lineGetNumAddressIDs	Retrieves the number of address identifiers supported on the indicated line.
lineHold	TSPI_lineHold	Places the specified call on hold.
lineMakeCall	TSPI_lineMakeCall	Places a call on the specified line to the specified destination address.
lineNegotiateExtVersion	TSPI_lineNegotiateExtVersion	Returns the highest extension version number the service provider can operate under for this device, given the range of possible extension versions.
	TSPI_lineNegotiateTSPIVersion	Returns the highest SPI version the service provider can operate under for this device, given the range of possible SPI versions.
lineOpen	TSPI_lineOpen	Opens the line device whose device identifier is given, returning the service provider's handle for the device.
lineSetCallParams	TSPI_lineSetCallParams	Sets certain parameters for an existing call.
	TSPI_lineSetDefaultMediaDetection	Tells the service provider the new set of media types to detect for the indicated line, replacing any previous set.
lineSetStatusMessages	TSPI_lineSetStatusMessages	Enables TAPI to specify which notification messages the service provider should generate for events related to status changes for the specified line or any of its addresses.
lineSetupTransfer	TSPI_lineSwapHold	Initiates a transfer of the call specified by *hdCall*.
lineUnhold	TSPI_lineUnhold	Retrieves the specified held call.

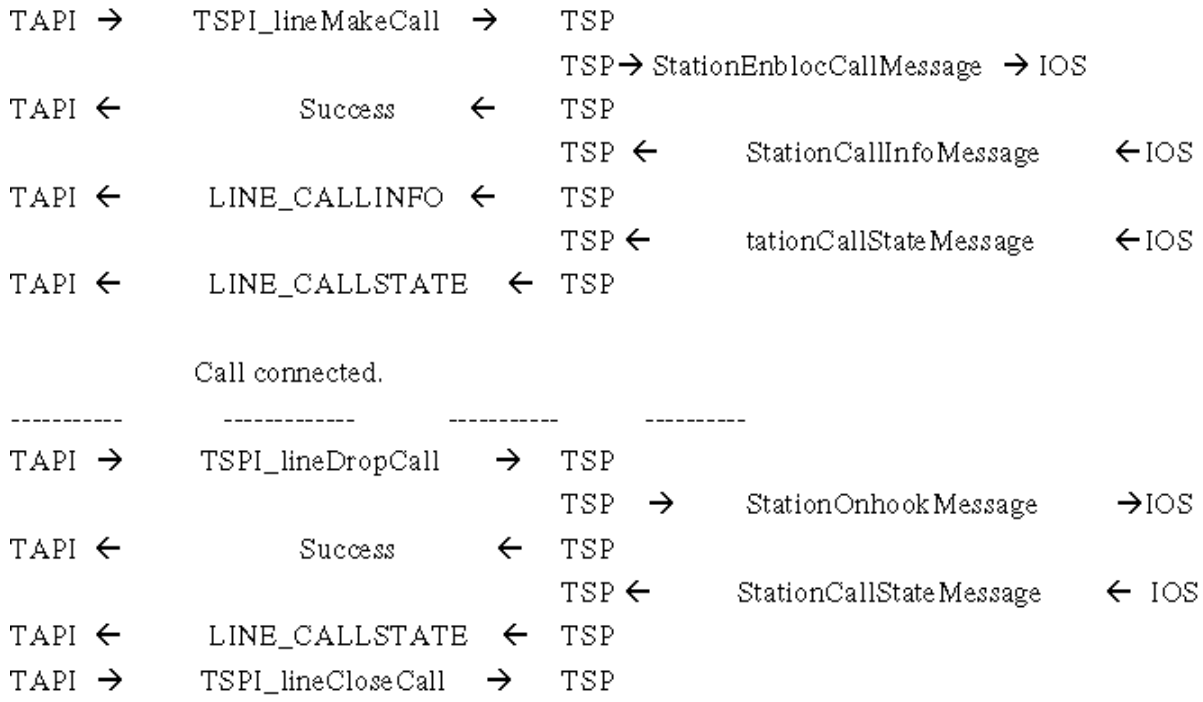
Flow Diagrams

This section presents examples of call flows:

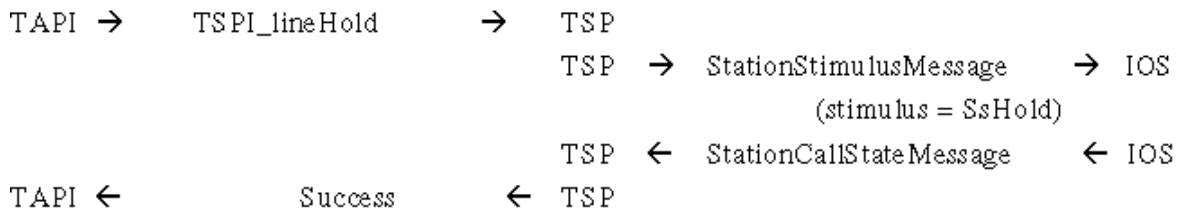
Initialization

TAPI → TSPI_lineNegotiateTSPIVersion → TSP
TAPI ← Success ← TSP
TAPI → TSPI_providerEnumDevices → TSP
TSP → StationConfigStatReqMessage → IOS
TSP ← StationConfigStatMessage ← IOS
TAPI ← Success ← TSP
TAPI → TSPI_providerInit → TSP
TSP → StationRegisterTokenMessage → IOS
(user id)
TSP → StationRegisterMessage → IOS
(Passwd).
TSP ← StationRegisterAckMessage ← IOS
TSP → StationButtonTemplateReq → IOS
TSP ← StationButtonTemplate ← IOS
TAPI ← Success ← TSP
TAPI → TSPI_lineNegotiateTSPIVersion → TSP
(for each line device)
TAPI ← Success ← TSP
TAPI. → TSPI_GetDevCaps → TSP
TAPI ← Success ← TSP
TAPI → TSPI_lineopen → TSP
TAPI ← Success ← TSP
TAPI → TSPI_GetDevStatus → TSP
TAPI ← Success ← TSP
TAPI → TSPI_GetAddressCaps → TSP
TAPI ← Success ← TSP
TAPI → TSPI_AddressStatus → TSP
TAPI ← Success ← TSP

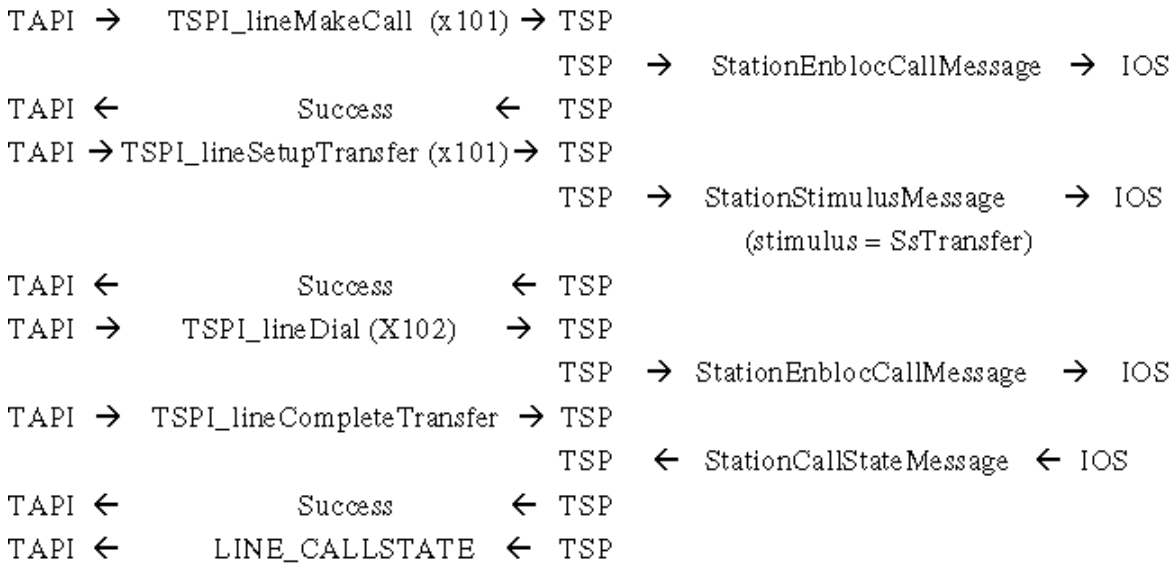
Application to Skinny IP Phone



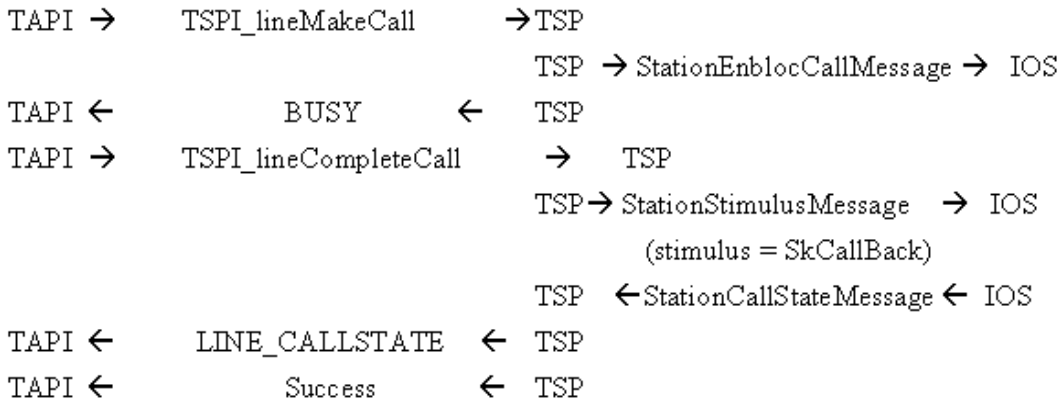
Hold Call



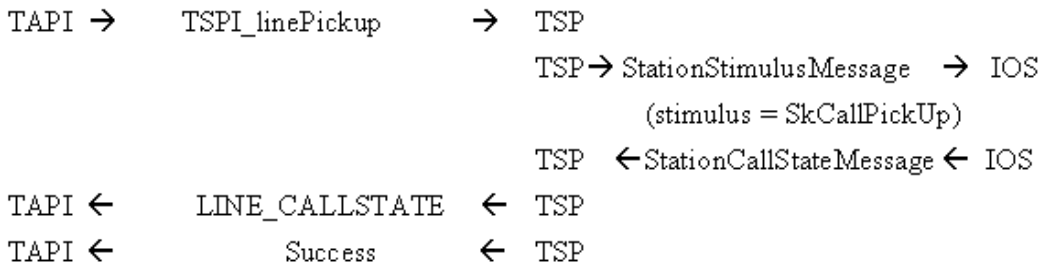
Call Transfer



Call Completion



Pickup Call



Park Call

TAPI → TSPI_linePark → TSP

TSP → StationStimulusMessage → IOS

(stimulus = SkPark)

TSP ← StationCallStateMessage ← IOS

TAPI ← LINE_CALLSTATE ← TSP

TAPI ← Success ← TSP

Conference Call

TAPI → TSPI_lineSetupConferenceCall (x101) → TSP

TSP → StationEnblocCallMessage → IOS

TAPI ← Success ← TSP

TAPI → TSPI_lineDial (X101) → TSP

TSP → StationEnblocCallMessage → IOS

TAPI → TSPI_lineAddToConference (x101) → TSP

TSP → StationStimulusMessage → IOS

(stimulus = SkConfrn)

TAPI ← Success ← TSP

TAPI → TSPI_linePrepareAddToConferenceCall (x102) → TSP

TSP → StationEnblocCallMessage → IOS

TAPI ← Success ← TSP

TAPI → TSPI_lineDial (X102) → TSP

TAPI → TSPI_lineAddToConference (x102) → TSP

TSP → StationStimulusMessage → IOS

(stimulus = SkConfrn)

TAPI ← Success ← TSP

TAPI ← LINE_CALLSTATE ← TSP

Appendix A

This section provides an quick reference for supported TSPI functions and messages. For complete information on TAPI and TSPI functions, see Microsoft™ TSPI reference.

This section is divided into the following areas:

- TSPI Line Functions
- TSPI Line Messages

TSPI Line Functions

This section provides an alphabetical listing of supported TSPI Line Functions.

TSPI_lineAddToConference

The **TSPI_lineAddToConference** function adds a call to a conference call. The call being added and the conference call are specified by *hdConfCall*.

Function Details

```
LONG TSPIAPI TSPI_lineAddToConference(
DRV_REQUESTID dwRequestID,
HDRVCALL hdConfCall,
HDRVCALL hdConsultCall );
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdConfCall*—The handle to the conference call. The call state of hdConfCall can be onHoldPendingConference or onHold.
- *hdConsultCall* —The handle to the call to be added to the conference call. This call cannot be either a parent of another conference or a participant in any conference.

  Depending on the device capabilities indicated in LINEADDRESSCAPS, the hdConsultCall parameter may not necessarily have been established using **TSPI_lineSetupConference** or **TSPI_linePrepareAddToConference**.

  The call state of *hdConsultCall* can be connected, onHold, proceeding, or ringback.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_CONFERENCEFULL
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM

Usage Notes

- The service provider returns LINEERR_INVALCALLHANDLE if *hdConsultCall* is a parent of another conference or already a participant in a conference.
- It also provides the same return value if *hdConsultCall* cannot be added or if it has been established using **TSPI_lineSetupConference** or **TSPI_linePrepareAddToConference**.
- Any monitoring (media, tones, digits) on a conference call applies only to the *hdConfCall* parameter, not to the individual participating calls.

TSPI_lineAnswer

The **TSPI_lineAnswer** function answers the specified offering call.

Function Details

```
LONG TSPIAPI TSPI_lineAnswer(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  LPCSTR lpsUserUserInfo,
  DWORD dwSize
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The service provider's handle to the call to be answered. The call state of *hdCall* can be *offering* or *accepted*.
- *lpsUserUserInfo*—A pointer to a null-terminated string containing user-user information to be sent to the remote party at the time of answering the call. If this pointer is NULL, it indicates that no user-user information is to be sent. User-user information is only sent if supported by the underlying network, as indicated in LINEDEVCAPS.
- *dwSize*—The size, in bytes, of the user-user information in *lpsUserUserInfo*. If *lpsUserUserInfo* is NULL, *dwSize* is ignored.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_INUSE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_USERUSERINFOTOOBIG.

Usage Notes

- When a new call arrives, the service provider sends TAPI a LINE_NEWCALL message to exchange handles for the call. The service provider follows this with a LINE_CALLSTATE message to inform TAPI and its client applications of the call's state. A client application typically answers the call using **TSPI_lineAnswer**. Typically, after the call is successfully answered, the call transitions to the *connected* state.
- In some telephony environments, like ISDN, where user alerting is separate from call offering, TAPI and its client applications may have the option to first accept a call prior to answering, or instead to reject or redirect the *offering* call.
- If a call is offered at the time another call is already active, the new call is connected to by invoking **TSPI_lineAnswer**. The effect this has on the existing active call depends on the line's device capabilities. The first call may be unaffected, it may automatically be dropped, or it may automatically be placed on hold. The appropriate LINE_CALLSTATE messages are used to report state transitions to TAPI about both calls.

TSPI_lineBlindTransfer

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

Function Details

```
LONG lineBlindTransfer(
HCALL hCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode
);
```

Parameters

- *hdCall*—The service provider's handle to the call to be transferred. The call state of *hdCall* can be *connected*.
- *lpszDestAddress*—A pointer to a null-terminated Unicode string identifying where the call is to be transferred to. The destination address uses the standard dialable number format.
- *dwCountryCode*—The country code of the destination. The implementation should use this to select the call progress protocols for the destination address. If a value of 0 is specified, the service provider should use a default. TAPI does not validate *dwCountryCode* when this function is called.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding *async_completion* is zero if the function succeeds or an error number if an error occurs.

Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_NOMEM
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESS
- LINEERR_OPERATIONFAILED
- LINEERR_ADDRESSBLOCKED

- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCOUNTRYCODE

Usage Notes

- The service provider carries out no dialing if it returns LINEERR_INVALADDRESS.
- In a blind transfer, no consultation call is made visible to TAPI. Typically, after the blind transfer is completed, the specified call is cleared from the line it was on and transitions to the *idle* state.
- The service provider call handle must remain valid after the transfer is completed. The handle becomes invalid when it is no longer interested in the transferred call using **TSPI_lineCloseCall**.

TSPI_lineCloseCall

The **TSPI_lineClose** function closes the specified open line device after completing or aborting all outstanding calls and asynchronous operations on the device.

Function Detail

```
LONG TSPIAPI TSPI_lineClose(
HDRVLINE hdLine
);
```

Parameters

- *hdLine*—The service provider's handle to the line to be closed. After the line is successfully closed, this handle is no longer valid.

Return Values

Returns zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_NOMEM
- LINEERR_OPERATIONFAILED
- LINEERR_OPERATIONUNAVAIL
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The service provider must report completion for every asynchronous operation. If **TSPI_lineClose** is called for a line on which there are outstanding asynchronous operations, the operations are reported complete with an appropriate result or error code before this procedure returns.
- A similar requirement exists for active calls on the line. Outstanding operations must be reported complete with appropriate result or error codes. Active calls must also be dropped, if required, and if this behavior was indicated by the LINEDEVCAPFLAGS_CLOSEDROP bit in the LINEDEVCAPS structure.
- After this procedure returns, the service provider must report no further *htCall* on the line or calls that were on the line. The service provider handles for the line and calls on the line become "invalid."
- The service provider must relinquish nonsharable resources it reserves while the line is open. For example, closing a line accessed through a comm port and modem should result in closing the comm port, making it once again available for use by other applications.
- The service provider does not issue the LINE_LINEDEVSTATE message in response to this function invocation because the line closes and there is no longer any interest in its state changes.

TSPI_lineCompleteCall

The **TSPI_lineCompleteCall** function specifies how to connect a call that cannot be normally completed.

Function Detail

```
LONG TSPIAPI TSPI_lineCompleteCall(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  LPDWORD lpdwCompletionID,
  DWORD dwCompletionMode,
  DWORD dwMessageID
);
```

Parameters

- dwRequestID—The identifier of the asynchronous request.
- hdCall—The service provider's handle to the call whose completion is requested. The call state of hdCall can be busy, ringback, or proceeding.
- lpdwCompletionID—A pointer to a DWORD-sized memory location where the service provider writes a completion identifier. This uniquely identifies a completion request in progress on the line that contains the hdCall. A completion identifier becomes invalid after the request completes or is canceled using the **TSPI_lineUncompleteCall** function. The service provider is free to reuse the completion identifier as soon as it becomes invalid.
- dwCompletionMode—The way in which the call is to be completed. This parameter uses only one of the LINECALLCOMPLMODE_ constants.
- 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. This parameter is not validated by TAPI when this function is called.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_NOMEM
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLCOMPLMODE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALPOINTER
- LINEERR_RESOURCEUNAVAIL
- LINEERR_COMPLETIONOVERRUN
- LINEERR_INVALMESSAGEID

Usage Notes

This function is considered complete when the request is accepted by the network or switch.

When the called station or network enters a state where the call can be completed as requested, the service provider must send a LINE_CALLSTATE message with the call state equal to offering. The call's LINECALLINFO record lists the reason for the call as CALLCOMPLETION and provides the completion identifier as well.

It is possible to have multiple call completion requests outstanding at any given time; the maximum number is device dependent. The completion identifier is also used to refer to each individual request so requests can be canceled by calling **TSPI_lineUncompleteCall**.

TSPI_lineCompleteTransfer

The **TSPI_lineCompleteTransfer** function completes the transfer of the specified call to the party connected in the consultation call. If *dwTransferMode* is LINETRANSFERMODE_CONFERENCE, the original call handle is changed to a conference call. Otherwise, the service provider sends call state messages changing the calls to *idle*.

Function Detail

```
LONG TSPIAPI TSPI_lineCompleteTransfer(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  HDRVCALL hdConsultCall,
  HTAPICALL htConfCall,
  LPHDRVCALL lphdConfCall,
  DWORD dwTransferMode
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The service provider's handle to the call to be transferred. The call state of *hdCall* can be *onHoldPendingTransfer*.
- *hdConsultCall*—A handle to the call that represents a connection to the destination of the transfer. The call state of *hdConsultCall* can be *connected*, *ringback*, *busy*, or *proceeding*.
- *htConfCall*—This parameter is only valid if *dwTransferMode* is specified as LINETRANSFERMODE_CONFERENCE. The service provider must save this parameter value and use it in all subsequent calls to the LINEEVENT procedure reporting events on the call. Otherwise this parameter is ignored.
- *lphdConfCall*—A pointer to an HDRVCALL representing the service provider's identifier for the call. This parameter is only valid if *dwTransferMode* is specified as LINETRANSFERMODE_CONFERENCE. The service provider must fill this location with its handle for the new conference call *before* returning from this function.
- *dwTransferMode*—Specifies how the initiated transfer request is to be resolved. This parameter uses one of the LINETRANSFERMODE_ constants.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL

- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- This function completes the transfer of the original call, *hdCall*, to the party currently connected through *hdConsultCall*. The consultation call is typically dialed on the consultation call allocated as part of **TSPI_lineSetupTransfer**, but it can be any call to which the switch is capable of transferring *hdCall*.
- The transfer request can be resolved either as a transfer or as a three-way conference call. When resolved as a transfer, the parties connected through *hdCall* and *hdConsultCall* are connected to each other, and both *hdCall* and *hdConsultCall* transition to the idle state.
- When resolved as a conference, all three parties enter into a conference call. Both existing call handles remain valid, but transition to the *conferenced* state. A conference call handle is created and returned, and it transitions to the *connected* state.
- It may also be possible to perform a blind transfer of a call using **TSPI_lineBlindTransfer**.
- This function differs from the corresponding TAPI function in that it follows the TSPI model for beginning the lifetime of a call. TAPI and the service provider exchange opaque handles representing the call with one another. In addition, the service provider is permitted to do callbacks for the new call before it returns from this procedure. In any case, the service provider must also treat the handle it returned as "not yet valid" until after the matching ASYNC_COMPLETION message reports success. In other words, it must not issue any LINEEVENT message for the new call or include it in call counts in messages or status data structures for the line.

TSPI_lineDial

The **TSPI_lineDial** function dials the specified dialable number on the specified call.

Function Detail

```
LONG TSPIAPI TSPI_lineDial(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  LPCWSTR lpszDestAddress,
  DWORD dwCountryCode
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The service provider's handle to the call to be dialed. The call state of *hdCall* can be any state except *idle* and *disconnected*.
- *lpszDestAddress*—Pointer to a null-terminated Unicode string that specifies the destination to be dialed using the standard dialable number format.
- *dwCountryCode*—The country code of the destination. The implementation uses this to select the call progress protocols for the destination address. If a value of 0 is specified, a default call-progress protocol defined by the service provider is used. TAPI does not validate this parameter when this function is called.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALADDRESS
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCOUNTRYCODE
- LINEERR_DIALBILLING
- LINEERR_INVALCALLSTATE
- LINEERR_DIALQUIET
- LINEERR_ADDRESSBLOCKED
- LINEERR_DIALDIALTONE
- LINEERR_NOMEM
- LINEERR_DIALPROMPT
- LINEERR_OPERATIONUNAVAIL

Usage Notes

- The service provider returns LINEERR_INVALCALLSTATE if the current state of the call does not allow dialing.
- The service provider carries out no dialing if it returns LINEERR_INVALADDRESS.
- If the service provider returns LINEERR_DIALBILLING, LINEERR_DIALQUIET, LINEERR_DIALDIALTONE, or LINEERR_DIALPROMPT, it should perform none of the actions otherwise performed by **TSPI_lineDial** (for example, no partial dialing, and no going offhook). This is because the service provider should pre-scan the number for unsupported characters first.
- **TSPI_lineDial** is used for dialing on an existing call appearance; for example, call handles returned from **TSPI_lineMakeCall** with NULL as the *lpszDestAddress* or ending in ';', call handles returned from **TSPI_lineSetupTransfer** or **TSPI_lineSetupConference**. **TSPI_lineDial** can be invoked multiple times in the course of dialing in the case of multistage dialing, if the line's device capabilities permit it.
- If the string pointed to by the *lpszDestAddress* parameter in the previous call to the **TSPI_lineMakeCall** or **TSPI_lineDial** function is terminated with a semicolon, an empty string in the current call to **TSPI_lineDial** indicates that dialing is complete.
- Multiple addresses can be provided in a single dial string separated by CRLF. Service providers that provide inverse multiplexing can establish individual physical calls with each of the addresses, and return a single call handle to the aggregate of all calls to the application. All addresses would use the same country code.
- Dialing is considered complete after the address has been accepted by the service provider, not after the call is finally connected. Service providers that provide inverse multiplexing may allow multiple addresses to be provided at once. The service provider must send LINE_CALLSTATE messages to TAPI to inform it about the progress of the call.

TSPI_lineDrop

The **TSPI_lineDrop** function drops or disconnects the specified call. User-user information can optionally be transmitted as part of the call disconnect. This function can be called by the application at any time. When **TSPI_lineDrop** returns, the call should be *idle*.

Function Detail

```
LONG TSPIAPI TSPI_lineDrop(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  LPCSTR lpsUserUserInfo,
  DWORD dwSize
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The service provider's handle to the call to be dropped. The call state of *hdCall* can be any state except *idle*.
- *lpsUserUserInfo*—This pointer is only valid if dwSize is nonzero. It specifies a pointer to a null-terminated string containing user-user information to be sent to the remote party as part of the call disconnect. This pointer is NULL if no user-user information is to be sent. User-user information is only sent if supported by the underlying network (see LINEDEVCAPS).
- *dwSize*—The size in bytes of the user-user information in *lpsUserUserInfo*. If *lpsUserUserInfo* is NULL, *dwSize* is ignored.

Return Values

Returns *dwRequestID* or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCALLSTATE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_USERUSERINFOTOOBIG
- LINEERR_OPERATIONUNAVAIL

Usage Notes

- The service provider returns LINEERR_INVALCALLSTATE if the current state of the call does not allow the call to be dropped.
- When invoking **TSPI_lineDrop**, related calls can sometimes be affected as well. For example, dropping a conference call may drop all individual participating calls. LINE_CALLSTATE messages are sent to TAPI for all calls whose call state is affected. Typically, a dropped call transitions to the *idle* state. Invoking **TSPI_lineDrop** on a call in the *offering* state rejects the call. Not all telephone networks provide this capability.
- In situations where the call to be dropped is a consultation call established during transfer or conference call establishment, the original call that was placed in the *OnHoldPending* state is reconnected to and it typically re-enters the *connected* call state.

- TAPI has the option to send user-user information at the time of the drop. Even if user-user information can be sent, there is no guarantee that the network will deliver this information to the remote party.

Note In various bridged or party line configurations when multiple parties are on the call, **TSPI_lineDrop** may not actually clear the call.

TSPI_lineGetAddressID

The **TSPI_lineGetAddressID** function returns the address identifier associated with address in a different format on the specified line.

Function Detail

```
LONG TSPIAPI TSPI_lineGetAddressID(
  HDRVLINE hdLine,
  LPDWORD lpdwAddressID,
  DWORD dwAddressMode,
  LPCWSTR lpsAddress,
  DWORD dwSize
);
```

Parameters

- *hdLine*—The service provider's handle to the line whose address is to be retrieved.
- *lpdwAddressID*—A pointer to a DWORD-sized memory location where the address identifier is returned.
- *dwAddressMode*—The address mode of the address contained in *lpsAddress*. The *dwAddressMode* parameter is allowed to have one and only one of the LINEADDRESSMODE_ constants.
- *lpsAddress*—A pointer to a data structure holding the address assigned to the specified line device. The format of the address is determined by the *dwAddressMode* parameter. If it is LINEADDRESSMODE_DIALABLEADDR, the *lpsAddress* parameter uses the common dialable number format and is NULL terminated.
- *dwSize*—The size of the address contained in *lpsAddress*. The parameter *dwSize* must be set to the length of the string (plus one for the NULL) if LINEADDRESSMODE_DIALABLEADDR is used. If an extended LINEADDRESSMODE is used, the length should match the size of whatever is actually passed in (the DLL checks to be sure it can read the number of bytes specified from the given pointer).

Return Values

Returns zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESS
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

This function is used to map a phone number (address) assigned to a line device back to its *dwAddressID* (in the range from 0 through the number of addresses minus one) that is returned in the line's device capabilities.

TSPI_lineGetCallAddressID

The **TSPI_lineGetCallAddressID** function retrieves the address identifier for the indicated call.

Function Detail

```
LONG TSPIAPI TSPI_lineGetCallAddressID(
  HDRVCALL hdCall,
  LPDWORD lpdwAddressID
);
```

Parameters

- *hdCall*—The service provider's handle to the call whose address identifier is to be retrieved. The call state of *hdCall* can be any state.
- *lpdwAddressID*—A pointer to a DWORD into which the service provider writes the call's address identifier.

Return Values

Returns zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_NOMEM
- LINEERR_OPERATIONFAILED
- LINEERR_OPERATIONUNAVAIL
- LINEERR_RESOURCEUNAVAIL

Remarks

- If the service provider models lines as "pools" of channel resources and does inverse multiplexing of a call over several address identifiers it should consistently choose one of these address identifiers as the primary identifier reported by this function and in the LINECALLINFO data structure.
- This function has no direct correspondence at the TAPI level. It gives TAPI sufficient information to implement the **lineGetNewCalls** function invoked with the LINECALLSELECT_ADDRESS option.

TSPI_lineGetCallInfo

The **TSPI_lineGetCallInfo** function returns detailed information about the specified call.

Function Detail

```
LONG TSPIAPI TSPI_lineGetCallInfo(
  HDRVCALL hdCall,
  LPLINECALLINFO lpCallInfo
);
```

Parameters

- *hdCall*—The service provider's handle to the call whose call information is to be retrieved. The call state of *hdCall* can be any state.
- *lpCallInfo*—A pointer to a variably sized data structure of type LINECALLINFO. Upon successful completion of the request, this structure is filled with call-related information.

Return Values

Returns zero if the function succeeds or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL
- LINEERR_OPERATIONUNAVAIL

Usage Notes

Table 2 indicates which members of the LINECALLINFO data structure are filled in by TAPI and which members are filled in by the service provider. The service provider must preserve (it must *not* overwrite) the values filled in by TAPI.

Table 2 LINECALLINFO Data Structure

Member name	TAPI	Service provider
dwTotalSize;	X	
dwNeededSize;		X
dwUsedSize;		X
hLine;	X	
dwLineDeviceID;		X
dwAddressID;		X
dwBearerMode;		X
dwRate;		X
dwMediaMode;		X
dwAppSpecific;		X
dwCallID;		X
dwRelatedCallID;		X
dwCallParamFlags;		X
dwCallStates;	X	X
dwMonitorDigitModes;	X	
dwMonitorMediaModes;	X	
DialParams;		X
dwOrigin;		X
dwReason;		X
dwCompletionID;		X

Table 2 LINECALLINFO Data Structure (continued)

Member name	TAPI	Service provider
dwNumOwners;	X	
dwNumMonitors;	X	
dwCountryCode;		X
dwTrunk;		X
dwCallerIDFlags;		X
dwCallerIDSize;		X
dwCallerIDOffset;		X
dwCallerIDNameSize;		X
dwCallerIDNameOffset;		X
dwCalledIDFlags;		X
dwCalledIDSize;		X
dwCalledIDOffset;		X
dwCalledIDNameSize;		X
dwCalledIDNameOffset;		X
dwConnectedIDFlags;		X
dwConnectedIDSize;		X
dwConnectedIDOffset;		X
dwConnectedIDNameSize;		X
dwConnectedIDNameOffset;		X
dwRedirectionIDFlags;		X
dwRedirectionIDSize;		X
dwRedirectionIDOffset;		X
dwRedirectionIDNameSize;		X
dwRedirectionIDNameOffset;		X
dwRedirectingIDFlags;		X
dwRedirectingIDSize;		X
dwRedirectingIDOffset;		X
dwRedirectingIDNameSize;		X
dwRedirectingIDNameOffset;		X
dwAppNameSize;	X	
dwAppNameOffset;	X	
dwDisplayableAddressSize;	X	
dwDisplayableAddressOffset;	X	
dwCalledPartySize;	X	
dwCalledPartyOffset;	X	
dwCommentSize;	X	

Table 2 LINECALLINFO Data Structure (continued)

Member name	TAPI	Service provider
dwCommentOffset;	X	
dwDisplaySize;		X
dwDisplayOffset;		X
dwUserUserInfoSize;		X
dwUserUserInfoOffset;		X
dwHighLevelCompSize;		X
dwHighLevelCompOffset;		X
dwLowLevelCompSize;		X
dwLowLevelCompOffset;		X
dwChargingInfoSize;		X
dwChargingInfoOffset;		X
dwTerminalModesSize;		X
dwTerminalModesOffset;		X
dwDevSpecificSize;		X
dwDevSpecificOffset;		X

TAPI fills in the size and offset fields for the dwAppNameSize/Offset, dwCalledPartySize/Offset, and dwCommentSize/Offset members and updates the value in dwUsedSize to reflect these after calling the service provider.

After the service provider returns from the **TSPI_lineGetCallInfo** function, TAPI sets the *dwCallStates* member of the LINECALLINFO structure as follows:

```
LINECALLINFO.dwCallStates |= LINECALLSTATE_UNKNOWN;
```

If the service provider models lines as "pools" of channel resources and does inverse multiplexing of a call over several address identifiers, it should consistently choose one of these address identifiers as the primary identifier reported by this function in the LINECALLINFO data structure.

TSPI_lineGetCallStatus

The **TSPI_lineGetCallStatus** function returns the current status of the specified call.

Function Detail

```
LONG TSPIAPI TSPI_lineGetCallStatus(
  HDRVCALL hdCall,
  LPLINECALLSTATUS lpCallStatus
);
```

Parameters

- *hdCall*—The service provider's handle to the call to be queried for its status. The call state of *hdCall* can be any state.
- *lpCallStatus*—A pointer to a variably sized data structure of type LINECALLSTATUS. This structure is filled with call status information.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL
- LINEERR_OPERATIONUNAVAIL

Usage Notes

- The following table indicates which members of the LINECALLSTATUS data structure are filled in by the service provider and which members are filled in by TAPI. The service provider must preserve (it must *not* overwrite) the values filled in by TAPI.

Table 3

Member name	TAPI	Service provider
dwTotalSize;	X	
dwNeededSize;		X
dwUsedSize;		X
dwCallState;		X
dwCallStateMode;		X
dwCallPrivilege;	X	
dwCallFeatures;		X
dwDevSpecificSize;		X
dwDevSpecificOffset;		X

- **TSPI_lineGetCallStatus** returns the dynamic status of a call, whereas **TSPI_lineGetCallInfo** returns primarily static information about a call. Call status information includes the current call state, detailed mode information related to the call while in this state (if any), as well as a list of the available TSPI functions TAPI can invoke on the call while the call is in this state.

TSPI_lineGetDevConfig

The **TSPI_lineGetDevConfig** function returns a data structure object, the contents of which are specific to the line (service provider) and device class, giving the current configuration of a device associated one-to-one with the line device.

Function Detail

```
LONG TSPIAPI TSPI_lineGetDevConfig(
  DWORD dwDeviceID,
  LPVARSTRING lpDeviceConfig,
  LPCWSTR lpszDeviceClass
);
```

Parameters

- *dwDeviceID*—The line device to be configured.
- *lpDeviceConfig*—A pointer to a data structure of type VARSTRING where the device configuration structure of the associated device is returned. Upon successful completion of the request, the service provider fills this data structure with the device configuration. The *dwStringFormat* member in the VARSTRING structure must be set to STRINGFORMAT_BINARY. If the dwTotalSize member of the VARSTRING structure pointed to by the *lpDeviceConfig* parameter is greater than or equal to the size of the fixed portion of the structure, the service provider sets the dwNeededSize member to the required size and returns zero.
- *lpszDeviceClass*—A pointer to a null-terminated Unicode string that specifies the device class of the device whose configuration is requested. Valid device class strings are the same as those specified for the **TSPI_lineGetID** function when it is applied to a line device (*dwSelect* has the value LINECALLSELECT_LINE).

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALDEVICECLASS
- LINEERR_NOMEM
- LINEERR_INVALPOINTER
- LINEERR_OPERATIONUNAVAIL
- LINEERR_STRUCTURETOOSMALL
- LINEERR_OPERATIONFAILED
- LINEERR_NODRIVER
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The call state is device-specific.
- This function can be used to retrieve a data structure from the service provider that specifies the configuration of a device associated one-to-one with the line device. The *lpszDeviceClass* parameter selects which of among possibly several different classes of devices is to have its configuration retrieved. The set of supported classes is restricted to those whose devices correspond one-to-one with the line device. For more information about common device classes, see TSPI Device Classes.
- A service provider should typically allow the TAPI/line device class under this function. It would retrieve parameters that have "line" scope, such as the list of addresses in this line, the list of physical hardware devices such as COMM ports corresponding to the addresses, maximum number of concurrent calls (if configurable), and so on.
- In general, this function does not allow media-related device classes such as mci waveaudio, low level wave, or datamodem device classes, because these usually apply to a particular call or a particular address. Because there can be more than one of these per line device, the identification of the particular call or address simply by the line device identifier parameter in this function would be ambiguous. An exception can be made for call-specific or address-specific device classes in cases where there is class configuration information that applies to the entire line device scope, such as initial defaults, and so on.

- There are several reasons why *exceptional* support for call-specific and address-specific device classes is of only limited value under this function. First, because these classes can be ambiguous on multiple-address/multiple-call service providers, only a subset of service providers support them. Applications are not likely to add a device-specific dependency on the inclusion of these classes in this function. Second, as higher-level media "classes" emerge that implement high-level protocols such as dial-in file system access in terms of low-level transport APIs, configuration for these classes tends toward "instance" scope instead of "class" scope. The high-level media API must supply its own functions to configure call-specific or address-specific instances.
- Whatever sort of devices and device classes this function supports, it can potentially affect two kinds of configuration information: permanent and temporary. *Permanent* information survives across different "opens" of the line, and even across different "inits" of the service provider itself. *Temporary* information survives only within a unique "open" of the line. When the line is closed, any such temporary information that has been retrieved or set through **TSPI_lineSetDevConfig** can revert to default or undefined values.

  The caller can reliably retrieve any temporary configuration only by a sequence such as **TSPI_lineOpen**, **TSPI_lineConfigDialog**, **TSPI_lineGetDevConfig**. The caller can reliably set temporary configuration information retrieved by such a sequence through a sequence such as **TSPI_lineOpen**, **TSPI_lineSetDevConfig**. The temporary part of configuration remains stable only until the next **TSPI_lineConfigDialog**, **TSPI_lineSetDevConfig**, or **TSPI_lineClose**. The service provider must take care of storing any permanent part of the configuration, typically in an .ini file, and reloading it whenever the service provider is initialized.
- The exact format of the data contained within the structure returned by this function is specific to the line and device class API, is undocumented, and is undefined. The structure returned by this function cannot be directly accessed or manipulated by the application, but can only be stored intact and later used in **TSPI_lineSetDevConfig** to restore the settings. The structure also cannot necessarily be passed to other devices, even of the same device class (although this may work in some instances, it is not guaranteed). A service provider should put items in the data structure to allow it to be checked for consistency to guard against failures due to a client application passing incompatible information.

TSPI_lineGetExtensionID

The **TSPI_lineGetExtensionID** function returns the extension identifier that the service provider supports for the indicated line device.

Function Detail

```
LONG TSPIAPI TSPI_lineGetExtensionID(
  DWORD dwDeviceID,
  DWORD dwTSPIVersion,
  LPLINEEXTENSIONID lpExtensionID
);
```

Parameters

- *dwDeviceID*—The line device to be queried.
- *dwTSPIVersion*—An interface version number that was already negotiated for this device using **TSPI_lineNegotiateTSPIVersion**. This function operates according to the interface specification at this version level.
- *lpExtensionID*—A pointer to a structure of type LINEEXTENSIONID. If the service provider supports provider-specific extensions, it fills this structure with the extension identifier of these extensions. If the service provider does not support extensions, it fills this structure with all zeros. (Therefore, a valid extension identifier cannot consist of all zeros.)

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_NOMEM,
- LINEERR_OPERATIONFAILED,
- LINEERR_OPERATIONUNAVAIL,
- LINEERR_RESOURCEUNAVAIL.

Usage Notes

- This function is typically called by TAPI in response to an application calling the lineNegotiateAPIVersion function. The result returned by the service provider should be appropriate for use in a subsequent call to **TSPI_lineNegotiateExtVersion**. An extension identifier of all zeros is not a legal extension identifier, because the all-zeros value is used to indicate that the service provider does not support extensions.

TSPI_lineGetID

The **TSPI_lineGetID** function returns a device identifier for the specified device class associated with the selected line, address, or call.

```
LONG TSPIAPI TSPI_lineGetID(
  HDRVLINE hdLine,
  DWORD dwAddressID,
  HDRVCALL hdCall,
  DWORD dwSelect,
  LPVARSTRING lpDeviceID,
  LPCWSTR lpszDeviceClass,
  HANDLE hTargetProcess
);
```

Parameters

- *hdLine*—The service provider's handle to the line to be queried.
- *dwAddressID*—An address on the given open line device. An address identifier is permanently associated with an address; the identifier remains constant across operating system upgrades. TAPI does not validate this parameter when this function is called.
- *hdCall*—The service provider's handle to the call to be queried.
- *dwSelect*—Specifies whether the device identifier requested is associated with the line, address, or a single call. The *dwSelect* parameter can have only one of the LINECALLSELECT_ constants.
- *lpDeviceID*—A pointer to the memory location of type VARSTRING where the device identifier is returned. Upon successful completion of the request, this location is filled with the device identifier. The format of the returned information depends on the method used by the device class (API) for naming devices.
- *lpszDeviceClass*—A pointer to a null-terminated Unicode string that specifies the device class of the device whose identifier is requested. Valid device class strings are those used in the System.ini section to identify device classes (such as COM, Wave, and MCI.)
- *hTargetProcess*—The process handle of the application on behalf of which the **TSPI_lineGetID** function is invoked. If the information returned in the VARSTRING structure includes a handle for use by the application, the service provider should create or duplicate the handle for the process.

If *hTargetProcess* is set to INVALID_HANDLE_VALUE, then the application is executing on a remote client system and it is not possible to create a duplicate handle directly. Instead, the VARSTRING structure should contain a UNC name of a network device or other name that the remote client can use to access the device. If this is not possible, then the function should fail.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_NOMEM
- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_NODEVICE
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The service provide returns LINEERR_INVALLINEHANDLE if *dwSelect* is LINECALLSELECT_LINE or LINECALLSELECT_ADDRESS, and *hdLine* is invalid.
- The service provider returns LINEERR_INVALCALLHANDLE if *dwSelect* is LINECALLSELECT_CALL and *hdCall* is invalid.
- The service provider should support the tapi/line device class to allow applications to determine the real line device identifier of an opened line. In that case, the variable data returned is the *dwDeviceID*. For more information about common device class names, see TSPI Device Classes.
- A vendor that defines a device-specific media type also needs to define the corresponding device-specific (proprietary) API to manage devices of the media type. To avoid collisions on device class names assigned independently by different vendors, a vendor should select a name that uniquely identifies the vendor and then the media type; for example: "intel/video".
- The service provider fills in all the members of the VARSTRING data structure, except for dwTotalSize, which is filled in by TAPI. The service provider must *not* overwrite the dwTotalSize member.
- The service provider does not need to be concerned with handling tapi/line and tapi/phone device classes because TAPI handles these for the service provider. Therefore, code for handling these device classes is optional.

TSPI_lineGetNumAddressIDs

The **TSPI_lineGetNumAddressIDs** function retrieves the number of address identifiers supported on the indicated line.

Function Detail

```
LONG TSPIAPI TSPI_lineGetNumAddressIDs(
  HDRVLINE hdLine,
  LPDWORD lpdwNumAddressIDs
);
```

Parameters

- *hdLine*—The handle to the line for which the number of address identifiers is to be retrieved.
- *lpdwNumAddressIDs*—A pointer to a DWORD. The location is filled with the number of address identifiers supported on the indicated line. The value is one or larger.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_NOMEM
- LINEERR_OPERATIONFAILED
- LINEERR_OPERATIONUNAVAIL
- LINEERR_RESOURCEUNAVAIL

Usage Notes

This function is called by TAPI in response to an application calling *lineSetNumRings*, *lineGetNumRings*, or *lineGetNewCalls*. TAPI uses the retrieved value to determine if the specified address identifier is within the range supported by the service provider.

TSPI_lineHold

The **TSPI_lineHold** function places the specified call on hold.

Function Detail

```
LONG TSPIAPI TSPI_lineHold(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The service provider's handle to the call to be placed on hold. The call state of *hdCall* can be *connected*.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The call on hold is temporarily disconnected, allowing TAPI to use the line device for making or answering other calls. **TSPI_lineHold** performs a *hard hold* of the specified call, as opposed to a *consultation call*. A call on hard hold typically cannot be transferred or included in a conference call, whereas a consultation call can. Consultation calls are initiated using **TSPI_lineSetupTransfer**, **TSPI_lineSetupConference**, or **TSPI_linePrepareAddToConference**.
- After a call is successfully placed on hold, the call state typically transitions to *onHold*. A held call is retrieved through **TSPI_lineUnhold**. While a call is on hold, the service provider can send LINE_CALLSTATE messages about state changes of the held call. For example, if the held party hangs up, the call state can transition to *disconnected*, and the service provider can send a LINE_CALLSTATE message indicating the new state.

TSPI_lineMakeCall

The **TSPI_lineMakeCall** function places a call on the specified line to the specified destination address. Optionally, call parameters can be specified if anything but default call setup parameters are requested.

Function Detail

```
LONG TSPIAPI TSPI_lineMakeCall(
  DRV_REQUESTID dwRequestID,
  HDRVLINE hdLine,
  HTAPICALL htCall,
  LPHDRVCALL lphdCall,
  LPCWSTR lpszDestAddress,
  DWORD dwCountryCode,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdLine*—The handle to the line on which the new call is to be originated.
- *htCall*—The TAPI handle to the new call. The service provider must save this and use it in all subsequent calls to the LINEEVENT procedure reporting events on the call.
- *lphdCall*—A pointer to a call handle. The service provider must fill this location with its handle for the call before this procedure returns. This handle is ignored by TAPI if the function results in an error.
- *lpszDestAddress*—Pointer to a null-terminated Unicode string that specifies the destination address. This follows the standard dialable number format. This pointer can be specified as NULL for non-dialed addresses (as with a hot phone, which always automatically connects to a predefined number) or when all dialing is performed using **TSPI_lineDial**.

  In the latter case, **TSPI_lineMakeCall** allocates an available call appearance that would typically remain in the dialtone state until dialing begins. Service providers that have inverse multiplexing capabilities can allow an application to specify multiple addresses at once.
- *dwCountryCode*—The country code of the called party. If a value of 0 is specified, a default is used by the implementation.
- *lpCallParams*—A pointer to a LINECALLPARAMS structure. This structure allows TAPI to specify how it wants the call to be set up. If NULL is specified, a default 3.1kHz voice call is established, and an arbitrary origination address on the line is selected. This structure selects elements such as the call's bearer mode, data rate, expected media type, origination address, blocking of caller ID information, and dialing parameters.

Return Values

Returns dwRequestID, or an error number if an error occurs. The lResult actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_ADDRESSBLOCKED
- LINEERR_INVALLINESTATE
- LINEERR_BEARERMODEUNAVAIL
- LINEERR_INVALRATE
- LINEERR_CALLUNAVAIL
- LINEERR_INVALLINEHANDLE
- LINEERR_DIALBILLING
- LINEERR_INVALADDRESS
- LINEERR_DIALQUIET
- LINEERR_INVALADDRESSID
- LINEERR_DIALDIALTONE
- LINEERR_INVALCALLPARAMS
- LINEERR_DIALPROMPT
- LINEERR_NOMEM
- LINEERR_INUSE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESSMODE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALBEARERMODE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCOUNTRYCODE
- LINEERR_RATEUNAVAIL
- LINEERR_INVALMEDIAMODE
- LINEERR_USERUSERINFOTOOBIG

Usage Notes

- The service provider returns LINEERR_INVALLINESTATE if the line is currently not in a state in which this operation can be performed. A list of currently valid operations can be found in the dwLineFeatures member (of the type LINEFEATURE) in the LINEDEVSTATUS structure. (Calling **TSPI_lineGetLineDevStatus** updates the information in LINEDEVSTATUS.)
- If the service provider returns LINEERR_DIALBILLING, LINEERR_DIALQUIET, LINEERR_DIALDIALTONE, or LINEERR_DIALPROMPT, it should perform none of the actions otherwise performed by **TSPI_lineMakeCall**. For example, no partial dialing, and no going offhook. This is because the service provider should pre-scan the number for unsupported characters first.

- After **TSPI_lineMakeCall** returns a SUCCESS reply callback message to the application, the service provider must send LINE_CALLSTATE messages to the *lpfnEventProc* passed in **TSPI_lineOpen** to notify TAPI about the progress of the call. A typical reported sequence may be dial tone, dialing, proceeding, ringback, and connected.

  The first state reported is not necessarily LINECALLSTATE_DIALTONE. The service provider chooses how many of these states are reported. It is recommended that as many as possible are sent, so that applications can take appropriate actions.
- The service provider initially does media monitoring on the new call for at least the set of media types that were monitored for on the line.
- If the dial string is NULL, the service provider takes the line to the dialtone state (for a Comm-based service provider, this would involve ATD) and send a call-state message indicating LINECALLSTATE_DIALTONE.
- If the dial string ends in ';' (a semicolon), the service provider dials the partial number and sends the LINECALLSTATE_DIALING message. This call is completed by calls to **TSPI_lineDial**.
- If the dial string contains characters (W, @, $, ?) that are not supported by the service provider, the service provider must scan the dial string and return (synchronously) an error corresponding to the first invalid character.
- If the LINECALLPARAMFLAGS_IDLE flag is set, the service provider must check the current line status (the equivalent of going off-hook and sensing dial tone). If this IDLE flag is set and there is no dial tone, the function fails with the error LINEERR_CALLUNAVAIL. If the IDLE flag is not set, or there is a dial tone, the dialing can continue.
- This function differs from the corresponding TAPI function in that it follows the TSPI model for beginning the lifetime of a call. TAPI and the service provider exchange opaque handles representing the call with one another. In addition, the service provider is permitted to do callbacks for the new call before it returns from this procedure. In any case, the service provider must also treat the handle it returned as "not yet valid" until after the matching ASYNC_COMPLETION message reports success. It must not issue any LINEEVENT messages for the new call or include it in call counts in messages or status data structures for the line.

TSPI_lineNegotiateExtVersion

The **TSPI_lineNegotiateExtVersion** function returns the highest extension version number the service provider can operate under for this device, given the range of possible extension versions.

Function Detail

```
LONG TSPIAPI TSPI_lineNegotiateExtVersion(
  DWORD dwDeviceID,
  DWORD dwTSPIVersion,
  DWORD dwLowVersion,
  DWORD dwHighVersion,
  LPDWORD lpdwExtVersion
);
```

Parameters

- *dwDeviceID*—Identifies the line device for which interface version negotiation is to be performed. The value INITIALIZE_NEGOTIATION may not be used for this function.
- *dwTSPIVersion*—An interface version number that has already been negotiated for this device using **TSPI_lineNegotiateTSPIVersion**. This function operates according to the interface specification at this version level.

- *dwLowVersion*—The lowest extension version number under which TAPI or its client application can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number. TAPI does not validate this parameter when this function is called.
- *dwHighVersion*—The highest extension version number under which TAPI or its client application can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number. TAPI does not validate this parameter when this function is called.
- *lpdwExtVersion*—A pointer to a DWORD. Upon a successful return from this function, the service provider fills this location with the highest extension version number, within the range requested by the caller, under which the service provider can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number. If the requested range does not overlap the range supported by the service provider, the function returns LINEERR_INCOMPATIBLEEXTVERSION.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INCOMPATIBLEEXTVERSION
- LINEERR_OPERATIONFAILED
- LINEERR_NODRIVER
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM

Usage Notes

This function can be called before or after the device is opened by TAPI. If the device is currently open and has an extension version selected, the function gives that version number if it is within the requested range. If the selected version number is outside the requested range, the function returns LINEERR_INCOMPATIBLEEXTVERSION.

TSPI_lineNegotiateTSPIVersion

The **TSPI_lineNegotiateTSPIVersion** function returns the highest SPI version the service provider can operate under for this device, given the range of possible SPI versions.

```
LONG TSPIAPI TSPI_lineNegotiateTSPIVersion(
  DWORD dwDeviceID,
  DWORD dwLowVersion,
  DWORD dwHighVersion,
  LPDWORD lpdwTSPIVersion
);
```

Parameters

- *dwDeviceID*—Identifies the line device for which interface version negotiation is to be performed. In addition to device identifiers within the range the service provider supports, this may be the value: INITIALIZE_NEGOTIATION. This value is used to signify that an overall interface version is to be negotiated.

- *dwLowVersion*—The lowest TSPI version number under which TAPI can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number.
- *dwHighVersion*—The highest TSPI version number under which TAPI can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number.
- *lpdwTSPIVersion*—A pointer to a DWORD. The service provider fills this location with the highest TSPI version number, within the range requested by the caller, under which the service provider can operate. The most-significant WORD is the major version number and the least-significant WORD is the minor version number. If the requested range does not overlap the range supported by the service provider, the function returns LINEERR_INCOMPATIBLEAPIVERSION.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INCOMPATIBLEAPIVERSION
- LINEERR_OPERATIONUNAVAIL
- LINEERR_NODRIVER
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

When *dwDeviceID* is INITIALIZE_NEGOTIATION, this function must not return LINEERR_OPERATIONUNAVAIL, because this function (with that value) is mandatory for negotiating the overall interface version even if the service provider supports no line devices.

TSPI_lineOpen

The **TSPI_lineOpen** function opens the line device whose device identifier is given, returning the service provider's handle for the device. The service provider must retain the TAPI handle for the device for use in subsequent calls to the LINEEVENT callback procedure.

Function Detail

```
LONG TSPIAPI TSPI_lineOpen(
  DWORD dwDeviceID,
  HTAPILINE htLine,
  LPHDRVLINE lphdLine,
  DWORD dwTSPIVersion,
  LINEEVENT lpfnEventProc
);
```

Parameters

- *dwDeviceID*—Identifies the line device to be opened.
- *htLine*—The TAPI handle for the line device to be used in subsequent calls to the LINEEVENT callback procedure to identify the device.
- *lphdLine*—A pointer to an HDRVLINE where the service provider fills in its handle for the line device.

- *dwTSPIVersion*—The TSPI version.
- *lpfnEventProc*—A pointer to the LINEEVENT callback procedure supplied by TAPI that the service provider calls to report subsequent events on the line.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_ALLOCATED
- LINEERR_OPERATIONUNAVAIL
- LINEERR_NODRIVER
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The service provider should reserve any non-sharable resources that are required to manage the line. However, any actions that can be postponed to lineMakeCall should be. It is a design assumption in TAPI that lineOpen is an "inexpensive" operation. For example, if the line is opened in monitor mode only, it should not be necessary for a COMM-port-based service provider to open the COMM port.
- This procedure does not correspond directly to any procedure at the TAPI level, at which the functions of enabling device-specific extensions, selecting line characteristics, and setting media type detection are included in the functionality defined by **lineOpen**. At the TSPI level, these additional capabilities are separated out into **TSPI_lineNegotiateExtVersion**, **TSPI_lineSetDefaultMediaDetection**, and **TSPI_lineConditionalMediaDetection**.

TSPI_linePark

The **TSPI_linePark** function parks the specified call according to the specified park mode.

Function Details

```
LONG TSPIAPI TSPI_linePark(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  DWORD dwParkMode,
  LPCWSTR lpszDirAddress,
  LPVARSTRING lpNonDirAddress
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The handle for the call being parked. The call state of *hdCall* is connected.
- *dwParkMode*—The park mode with which the call is to be parked. It is only one of the LINEPARKMODE_ constants.
- *lpszDirAddress*—A pointer toa null-terminated Unicode string that indicates the address where the call is to be parked when using directed park. This parameter is ignored for nondirected park.

- *lpNonDirAddress*—A 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 containing a null-terminated string), and the terminating NULL is accounted for in the *dwStringSize*.

  If the memory pointed to by the *lpNonDirAddress* parameter is not large enough for the requested address, the **TSPI_linePark** function returns LINEERR_STRUCTURETOOSMALL.

Return Values

If an error occurs, an error number or *dwRequestID* is returned. The *lResult* parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_NOMEM
- LINEERR_INVALPARKMODE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALADDRESS
- LINEERR_RESOURCEUNAVAIL
- LINEERR_STRUCTURETOOSMALL

Usage Notes

A parked call enters an idle state after it is successfully parked. The service provider reports the new state using a LINE_CALLSTATE message. A subsequent **TSPI_lineUnpark** creates a new distinct call handle, regardless of whether **TSPI_lineCloseCall** was used to destroy the old handle.

TSPI_linePickup

The **TSPI_linePickup** function picks up a call alert 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 capabilities, *lpszGroupID* specifies the group identifier to which the alerting station belongs.

Function Details

```
LONG TSPIAPI TSPI_linePickup(
  DRV_REQUESTID dwRequestID,
  HDRVLINE hdLine,
  DWORD dwAddressID,
  HTAPICALL htCall,
  LPHDRVCALL lphdCall,
  LPCWSTR lpszDestAddress,
  LPCWSTR lpszGroupID
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.

- *hdLine*—The handle to the line on which a call is to be picked up.
- *dwAddressID*—The address on hdLine 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.
- *htCall*—The TAPI handle to the new call. The service provider must save this and use it in all subsequent calls to the LINEEVENT procedure reporting events on the call.
- *lphdCall*—A pointer to an HDRVCALL representing the service provider's identifier for the call. The service provider must fill this location with its handle for the call before this procedure returns. This handle is ignored by TAPI if the function results in an error.
- *lpszDestAddress*—A pointer to a null-terminated Unicode string that contains the address whose call is to be picked up. The address is standard link format.
- *lpszGroupID*—A pointer to a null-terminated Unicode string 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.

  lpszGroupID 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. It can also be NULL itself.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_NOMEM
- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESS
- LINEERR_OPERATIONFAILED
- LINEERR_INVALGROUPID
- LINEERR_RESOURCEUNAVAIL

Usage Notes

By calling **TSPI_lineGetCallInfo** when a call has been picked up successfully, the service provider notifies TAPI with the LINE_CALLSTATE message about call state changes. The LINECALLINFO structure supplies information about the call that was picked up and lists the reason for the pickup.

TSPI_linePrepareAddToConference

The **TSPI_linePrepareAddToConference** function creates a consultation call that can be added to an existing conference call.

Function Details

```
LONG TSPIAPI TSPI_linePrepareAddToConference(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdConfCall,
  HTAPICALL htConsultCall,
```

```
  LPHDRVCALL lphdConsultCall,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdConfCall*—The handle to a conference call. The call state of hdConfCall can be connected.
- *htConsultCall*—The TAPI handle to the new consultation call. The service provider must save this and use it in all subsequent calls to the LINEEVENT procedure reporting events on the new call. The call state of hdAddCall is not applicable.
- *lphdConsultCall*—A pointer to an HDRVCALL representing the service provider's identifier for the new consultation call. The service provider must fill this location with its handle for the new call before this procedure returns. This handle is invalid if the function results in an error.
- *lpCallParams*—A pointer to a LINECALLPARAMS containing call parameters to use when establishing the consultation call. This parameter is set to NULL if no special call setup parameters are desired.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_BEARERMODEUNAVAIL
- LINEERR_INVALLINESTATE
- LINEERR_CALLUNAVAIL
- LINEERR_INVALMEDIAMODE
- LINEERR_CONFERENCEFULL
- LINEERR_INVALRATE
- LINEERR_INUSE
- LINEERR_NOMEM
- LINEERR_INVALADDRESSMODE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALBEARERMODE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALCALLPARAMS
- LINEERR_RATEUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_INVALCONFCALLHANDLE
- LINEERR_USERUSERINFOTOOBIG

Usage Notes

This function places an existing conference call on hold (onHoldPendingConference) and creates a consultation call that can be added later to the existing conference call with **TSPI_lineAddToConference**.

TSPI_lineSetCallParams

The **TSPI_lineSetCallParams** function sets certain parameters for an existing call.

Function Detail

```
LONG TSPIAPI TSPI_lineSetCallParams(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  DWORD dwBearerMode,
  DWORD dwMinRate,
  DWORD dwMaxRate,
  LPLINEDIALPARAMS const lpDialParams
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The handle to the call whose parameters are to be changed. The call state can be any state except *idle* and *disconnected*.
- *dwBearerMode*—The new bearer mode for the call. The *dwBearerMode* parameter can have only one of the LINEBEARERMODE_ constants.
- *dwMinRate*—A lower bound for the call's new data rate. TAPI can accept a new rate as low as this one. TAPI does not validate this parameter when this function is called.
- *dwMaxRate*—An upper bound for the call's new data rate. This is the maximum data rate TAPI would like. Equal values for *dwMinRate* and *dwMaxRate* indicate that an exact data rate is required. TAPI does not validate this parameter when this function is called.
- *lpDialParams*—A pointer to the new dial parameters for the call, of type LINEDIALPARAMS. If this parameter is NULL, it indicates that the call's current dialing parameters are to be used.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_RATEUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_NOMEM
- LINEERR_INVALBEARERMODE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALPOINTER
- LINEERR_OPERATIONFAILED
- LINEERR_INVALRATE

- LINEERR_RESOURCEUNAVAIL
- LINEERR_BEARERMODEUNAVAIL

Usage Notes

This operation is used to change the parameters of an existing call. Examples of its usage include changing the bearer mode or the data rate of an existing call.

TSPI_lineSetupTransfer

The **TSPI_lineSetupTransfer** function initiates a transfer of a call specified by *hdCall*.

Function Details

```
LONG TSPIAPI TSPI_lineSetupTransfer(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall,
  HTAPICALL htConsultCall,
  LPHDRVCALL lphdConsultCall,
  LPLINECALLPARAMS const lpCallParams
);
```

Parameters

dwRequestID—The identifier of the asynchronous request.

hdCall—The handle to the call to be transferred. The call state of hdCall can be connected.

htConsultCall—The TAPI handle to the new consultation call. The service provider must save this and use it in all subsequent calls to the LINEEVENT procedure reporting events on the new consultation call.

lphdConsultCall—A pointer to an HDRVCALL representing the service provider's identifier for the new consultation call.

lpCallParams—A pointer to a LINECALLPARAMS structure containing call parameters to use when establishing the consultation call. This parameter can be set to NULL if no special call setup parameters are desired (the service provider uses defaults).

Return Values

Returns dwRequestID, or an error number if an error occurs. The lResult parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_INVALBEARERMODE
- LINEERR_INVALCALLSTATE
- LINEERR_INVALRATE
- LINEERR_CALLUNAVAIL
- LINEERR_INVALCALLPARAMS
- LINEERR_NOMEM
- LINEERR_INVALLINESTATE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALMEDIAMODE
- LINEERR_OPERATIONFAILED

- LINEERR_INUSE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_BEARERMODEUNAVAIL
- LINEERR_RATEUNAVAIL
- LINEERR_INVALADDRESSMODE
- LINEERR_USERUSERINFOTOOBIG

Usage Notes

This operation sets up the transfer of the call specified by *hdCall*. The setup phase of the transfer establishes a new call that sends the address of the destination party to the switch, while the call to be transferred is kept on hold.

This new call is referred to as a consultation call (*hdConsultCall*) and can be manipulated (for example, dropped) independently of the original call.

TSPI_lineSetDefaultMediaDetection

The **TSPI_lineSetDefaultMediaDetection** function tells the service provider the new set of media types to detect for the indicated line (replacing any previous set). It also sets the initial set of media types that should be monitored for on subsequent calls (inbound or outbound) on this line.

Function Detail

```
LONG TSPIAPI TSPI_lineSetDefaultMediaDetection(
  HDRVLINE hdLine,
  DWORD dwMediaModes
);
```

Parameters

- *hdLine*—The handle to the line to have media monitoring set.
- *dwMediaModes*—The media type(s) of interest to TAPI. This parameter uses one of the LINEMEDIAMODE_ constants:

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALMEDIAMODE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM
- LINEERR_NODRIVER
- LINEERR_OPERATIONUNAVAIL

Usage Notes

- TAPI typically calls this function to update the set of detected media types for the line to the union of all modes selected by all outstanding lineOpens whenever a line is Opened or Closed at the TAPI level. A lineOpen call attempt is rejected if media detection is rejected. A single call to this procedure is typically the result of a lineOpen call that does not specify the device identifier LINEMAPPER. The device identifier LINEMAPPER is never used at the TSPI level.
- TAPI must compute the union of media types desired by all clients and pass the result to this function. The service provider uses the set passed to this function by TAPI. TAPI ensures that the *dwMediaModes* parameter has at least one bit set and that no reserved bits are set.
- The union of all media types can be the value 0 if the applications that have the line open are all either monitors or not interested in handling incoming calls.
- There is no directly corresponding function at the TAPI level. This procedure corresponds to the "request media types" implied for the specific line by the lineOpen procedure when it is called with the specific device identifier (other than LINEMAPPER).

TSPI_lineSetStatusMessages

The **TSPI_lineSetStatusMessages** function enables TAPI to specify which notification messages the service provider should generate for events related to status changes for the specified line or any of its addresses.

Function Detail

```
LONG TSPIAPI TSPI_lineSetStatusMessages(
  HDRVLINE hdLine,
  DWORD dwLineStates,
  DWORD dwAddressStates
);
```

Parameters

- *hdLine*—The handle to the line device for which the new filter is to be set.
- *dwLineStates*—A bit array that identifies for which line device status changes a message is to be sent to TAPI. This parameter uses one of the LINEDEVSTATE_ constants.
- *dwAddressStates*—A bit array that identifies for which address status changes a message is to be sent to TAPI. This parameter uses one of the LINEADDRESSSTATE_ constants.

Return Values

Returns zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALADDRESSSTATE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONFAILED
- LINEERR_INVALLINESTATE
- LINEERR_RESOURCEUNAVAIL
- LINEERR_NOMEM

Usage Notes

- The service provider returns LINEERR_INVALLINESTATE if the *dwLineStates* parameter contains one or more bits that are not LINEDEVSTATE_ constants.
- Telephony defines a number of messages that notify applications about events occurring on lines and addresses. The sets of all change messages in which all applications are interested can be much smaller than the set of possible messages. This procedure allows TAPI to tell the service provider the reduced set of messages to deliver. The service provider delivers all of the messages it supports, within the specified set. It is permitted to deliver more (they are filtered out by TAPI), but is discouraged from doing so for performance reasons.

  If TAPI requests delivery of a particular message type that is not produced by the provider, the provider nevertheless accepts the request but simply does not produce the message. By default, address and line status reporting is initially disabled for a line.
- This function differs from the corresponding TAPI function as follows: (1) The set of messages requested is the union of all sets requested by applications at the TAPI level. (2) The message set is neither reduced nor augmented by ownership (because there is no concept of ownership at the TSPI level) (3) The set is advisory in the sense that the service provider is required to forward at least the indicated set of messages but is permitted to forward a larger set.
- Device state changes regarding Open and Close are not reported, because at the TSPI level there is only one outstanding Open at a time.

TSPI_lineSwapHold

The **TSPI_lineSwapHold** function swaps the active call with the consultation call that is placed on hold.

Function Details

```
LONG TSPIAPI TSPI_lineSwapHold(
DRV_REQUESTID dwRequestID,
HDRVCALL hdActiveCall,
HDRVCALL hdHeldCall );
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdActiveCall*—The handle to the call to be swapped with the call on consultation hold. The call state of *hdActiveCall* can be connected.
- *hdHeldCall*—The handle to the consultation call. The call state of *hdHeldCall* can be onHoldPendingTransfer, onHoldPendingConference, or onHold.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs.

Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

The service provider sends LINE_CALLSTATE messages for all call transitions.

TSPI_lineUnhold

The **TSPI_lineUnhold** function retrieves the call on hold.

Function Detail

```
LONG TSPIAPI TSPI_lineUnhold(
  DRV_REQUESTID dwRequestID,
  HDRVCALL hdCall
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdCall*—The handle to the call to be retrieved. The call state of *hdCall* can be *onHold*.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The *lResult* actual parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALCALLHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCALLSTATE
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- The service provider returns LINEERR_INVALCALLSTATE if the call is not currently on hold.
- This operation works for calls on hard hold (calls placed on hold using **TSPI_lineHold**) and on soft hold. The service provider should check that the call is currently in the *onHold*, *onHoldPendingTransfer*, or *onHoldPendingConference* state, change the state to *connected*, and send a LINECALLSTATE message for the new call state.

TSPI_lineUncompleteCall

The **TSPI_lineUncompleteCall** function is used to cancel the call completion request on the specified line.

Function Details

```
LONG TSPIAPI TSPI_lineUncompleteCall(
  DRV_REQUESTID dwRequestID,
  HDRVLINE hdLine,
  DWORD dwCompletionID
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdLine*—The handle to the line on which a call completion is to be canceled.
- *dwCompletionID*—The completion identifier for the request that is to be canceled.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The lResult parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALCOMPLETIONID
- LINEERR_OPERATIONFAILED
- LINEERR_NOMEM
- LINEERR_RESOURCEUNAVAIL

TSPI_lineUnpark

The **TSPI_lineUnpark** function retrieves the call parked at a specified address and returns the call handle for it.

Function Details

```
LONG TSPIAPI TSPI_lineUnpark(
  DRV_REQUESTID dwRequestID,
  HDRVLINE hdLine,
  DWORD dwAddressID,
  HTAPICALL htCall,
  LPHDRVCALL lphdCall,
  LPCWSTR lpszDestAddress
);
```

Parameters

- *dwRequestID*—The identifier of the asynchronous request.
- *hdLine*—The handle to the line on which a call is to be unparked.
- *dwAddressID*—The address on *hdLine* at which **TSPI_lineUnpark** originates.
- *htCall*—The TAPI handle to the new unparked call. The service provider must save this and use it in all subsequent calls to the LINEEVENT procedure reporting events on the call.
- *lphdCall*—A pointer to an HDRVCALL representing the service provider's identifier for the new unparked call. The service provider must fill this location with its handle for the call before this procedure returns. This handle is invalid if the function results in an error.
- *lpszDestAddress*—A pointer to a null-terminated Unicode string that contains the address where the call is parked. The address is in dialable address format.

Return Values

Returns *dwRequestID*, or an error number if an error occurs. The lResult parameter of the corresponding ASYNC_COMPLETION is zero if the function succeeds, or an error number if an error occurs. Possible return values are as follows:

- LINEERR_INVALLINEHANDLE
- LINEERR_NOMEM
- LINEERR_INVALPOINTER
- LINEERR_OPERATIONUNAVAIL
- LINEERR_INVALADDRESSID
- LINEERR_OPERATIONFAILED
- LINEERR_INVALADDRESS
- LINEERR_RESOURCEUNAVAIL

Usage Notes

- This function follows the TSPI model for beginning the lifetime of a call. TAPI and the service provider exchange opaque handles representing the call with one another.
- The service provider is permitted to do callbacks for the new call before it returns from this procedure.
- The service provider must also treat the handle it returned as invalid until the matching SYNC_COMPLETION message reports success. It must not issue any LINEEVENT messages for the new call or include it in call counts in messages or status data structures for the line.
- The call handle created by this function is a new, distinct, call handle even if an original call handle for the call is still in existence.

TSPI Line Messages

This section provides an alphabetical listing of supported TAPI Line Messages.

Table 4 TAPI Line Messages at a Glance

Function	Description
LINE_ADDRESSSTATE	Sent to the LINEEVENT callback function when the status of an address changes on a line that is currently open.
LINE_CALLINFO	Sent to the LINEEVENT callback function when the call information about the specified call has changed.
LINE_CALLSTATE	Sent to the LINEEVENT callback function whenever the status of the specified call has changed.
LINE_CLOSE	Sent to the LINEEVENT callback function by the service provider to request that TAPI close the specified line device.
LINE_CREATEDIALOGINSTANCE	Causes TAPI to create an association between the service provider and an instance of the TUISPI_providerGenericDialog function
LINE_LINEDEVSTATE	Sent to the LINEEVENT callback function when the state of a line device has changed.

Table 4 TAPI Line Messages at a Glance

Function	Description
LINE_NEWCALL	Sent to the LINEEVENT callback function whenever a new call that TAPI has not originated arrives on a line that TAPI has open.
LINE_SENDDIALOGINSTANCEDATA	Causes TAPI to call the TUISPI_providerGenericDialogData function in the UI DLL associated with htDlgInst

LINE_ADDRESSSTATE

The TSPI **LINE_ADDRESSSTATE** message is sent to the LINEEVENT callback function when the status of an address changes on a line that is currently open by TAPI. TAPI can invoke **TSPI_lineGetAddressStatus** to determine the current status of the address.

Message Detail

```
LINE_ADDRESSSTATE
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) 0;
dwMsg = (DWORD) LINE_ADDRESSSTATE;
dwParam1 = (DWORD) idAddress;
dwParam2 = (DWORD) AddressState;
dwParam3 = (DWORD) 0;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—Unused.
- *dwMsg*—The value LINE_ADDRESSSTATE.
- *dwParam1*—The address identifier of the address that changed status.
- *dwParam2*—The address state that changed. This parameter can be a combination of the LINEADDRESSSTATE_ constants.
- *dwParam3*—Unused.

Usage Notes

- This message is sent whenever the line is open by TAPI and an event occurs in which TAPI has expressed an interest. TAPI uses the **TSPI_lineSetStatusMessages** function to specify the set of status-change events in which it is interested. By default, address status reporting is disabled.
- For backward compatibility, older service providers are not expected to generate this value in a LINE_ADDRESSSTATE message. If they do, the message should be handled in the same manner as for newer service providers (as described earlier).

LINE_CALLINFO

The TSPI **LINE_CALLINFO** message is sent to the LINEEVENT callback function when the call information about the specified call has changed. TAPI can invoke **lineGetCallInfo** to determine the current call information.

Message Detail

```
LINE_CALLINFO
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) hCallDevice;
dwMsg = (DWORD) LINE_CALLINFO;
dwParam1 = (DWORD) CallInfoState;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—The TAPI opaque object handle to the call device.
- *dwMsg*—The value LINE_CALLINFO.
- *dwParam1*—Specifies the call information item that has changed. This parameter can be a combination of the LINECALLINFOSTATE_ constants.
- *dwParam2*—Unused.
- *dwParam3*—Unused.

Usage Notes

This message is sent to TAPI whenever the event occurs and TAPI has the line open. However, no LINE_CALLINFO messages are sent for a call after the call has entered the *idle* state.

LINE_CALLSTATE

The TSPI **LINE_CALLSTATE** message is sent to the LINEEVENT callback function whenever the status of the specified call has changed. Several such messages are typically sent during the lifetime of a call. The first such message for an incoming call indicates the offering state. TAPI can use **TSPI_lineGetCallStatus** to find out more detailed information about the current status of the call.

Message Detail

```
LINE_CALLSTATE
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) hCallDevice;
dwMsg = (DWORD) LINE_CALLSTATE;
dwParam1 = (DWORD) LineCallState;
dwParam2 = (DWORD) StateData;
dwParam3 = (DWORD) MediaMode;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—The TAPI opaque object handle to the call device.
- *dwMsg*—The value LINE_CALLSTATE.
- *dwParam1*—The new call state. This parameter is one of the LINECALLSTATE_ constants.
- *dwParam2*—Specifies call-state-dependent information.
  - If dwParam1 is LINECALLSTATE_BUSY, the dwParam2 parameter contains the details about the busy mode, and uses the LINEBUSYMODE_ constants.
  - If dwParam1 is LINECALLSTATE_DIALTONE, the dwParam2 parameter contains the details about the dial tone mode, and uses the LINEDIALTONEMODE_ constants.

  - If dwParam1 is LINECALLSTATE_SPECIALINFO, the dwParam2 parameter contains the details about the special info mode and uses the LINESPECIALINFO_ constants.
  - If dwParam1 is LINECALLSTATE_DISCONNECTED, the dwParam2 parameter contains the details about the disconnect mode, and uses the LINEDISCONNECTMODE_ constants.
  - If dwParam1 is LINECALLSTATE_CONFERENCED, dwParam2 contains the htCall of the parent call of the conference of which the subject htCall is a member. If the call specified in dwParam2 was not previously considered by TAPI to be a parent conference call, this message causes it to be so treated. The call specified in dwParam2 must already exist; it was most likely previously created by a LINE_NEWCALL message and set to LINECALLSTATE_ONHOLDPENDCONF.

- *dwParam3*—The media type of the call, as far as it is known. This is a combination of one or more LINEMEDIAMODE_ constants. If the service provider does not know the media type, it should include the "UNKNOWN" bit with all media types currently being monitored for.

Usage Notes

- The LINE_CALLSTATE message (with LINECALLSTATE_OFFERING) should be sent as the next message for an incoming call after LINE_NEWCALL. Other call state changes are reported whenever they occur; the message cannot be disabled.
- The LINE_CALLSTATE message also notifies TAPI about the existence and state of outbound calls established as a side effect of other calls (for example, when an active call is put on hold and replaced by a new call in the dialtone state) or manually by the user (for example, on an attached phone device). The call state of such calls reflects the actual state of the call, which is not offering. By examining the call state, TAPI can determine whether the call is an inbound call that needs to be answered.
- The corresponding message at the TAPI level is used to inform applications of new incoming calls. This is not the case at the TSPI level; the LINE_NEWCALL message informs TAPI of new incoming calls. The LINE_NEWCALL message must precede this message.
- The dwParam3 parameter is used at the TAPI level to inform the recipient of the privilege level it has over the call.
- For backward compatibility, older service providers do not pass a valid htCall in dwParam2. TAPI must check the value passed, and ignore it if it is not a valid htCall. If the value is a valid htCall, TAPI also checks the API version in use on the line device, and establishes a conference call internally only if the API version is 1.4 or later (for example, if the API version on the line is later than 1.4, this parameter should be ignored).

LINE_CLOSE

The TSPI **LINE_CLOSE** message is sent to the LINEEVENT callback function by the service provider to request that TAPI close the specified line device. Once closed, the line device handle or any call handles for calls on the line are no longer valid. The service provider guarantees that all asynchronous requests on the line and all calls on the line have been reported complete by calling ASYNC_COMPLETION for each outstanding request before this message is sent. TAPI must not request any future operations using this line handle or its associated call handles.

Message Detail

```
LINE_CLOSE
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) 0;
dwMsg = (DWORD) LINE_CLOSE;
dwParam1 = (DWORD) 0;
```

```
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—Unused.
- *dwMsg*—The value LINE_CLOSE.
- *dwParam1*—Unused.
- *dwParam2*—Unused.
- *dwParam3*—Unused.

Usage Notes

- The LINE_CLOSE message is a *request* from the service provider to TAPI to close the line. The service provider must not actually close the line until TAPI acknowledges the request by calling the **TSPI_lineClose** function to direct the service provider to close the line. A service provider can send this message, for example, when taking the line out of service or reconfiguring the service provider. Whether or not the line can be reopened immediately after it is closed is up to the service provider.
- A service provider can also request that a line device be closed after the user has modified the configuration of that line or its driver. If the user wants the configuration changes to be effective immediately (as opposed to after the next system restart), and they affect the application's current view of the device (such as a change in device capabilities), a service provider can close the line device.

LINE_CREATEDIALOGINSTANCE

The TSPI **LINE_CREATEDIALOGINSTANCE** message causes TAPI to create an association between the service provider and an instance of the TUISPI_providerGenericDialog function executing in the context of the application that invoked the asynchronous TSPI function identified by the *dwRequestID* parameter of the TUISPICREATEDIALOGINSTANCEPARAMS structure pointed to by *lpTUISPICreateDialogInstanceParams*.

Message Detail

```
LINE_CREATEDIALOGINSTANCE
htLine = (DWORD) hProvider;
htCall = (DWORD) 0;
dwMsg = LINE_CREATEDIALOGINSTANCE;
dwParam1 = (DWORD) lpTUISPICreateDialogInstanceParams;
dwParam2 = (DWORD) 0;
dwParam3 = (DWORD) 0;
```

Parameters

- *hProvider*—The ProviderHandle supplied to the service provider as a parameter to **TSPI_providerEnumDevices**.
- *lpTUISPICreateDialogInstanceParams*—Pointer to a TUISPICREATEDIALOGINSTANCEPARAMS structure.

LINE_LINEDEVSTATE

The TSPI **LINE_LINEDEVSTATE** message is sent to the LINEEVENT callback function when the state of a line device has changed. TAPI can invoke **TSPI_lineGetLineDevStatus** to determine the new status of the line.

Message Detail

```
LINE_LINEDEVSTATE
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) 0;
dwMsg = (DWORD) LINE_LINEDEVSTATE;
dwParam1 = (DWORD) LineDevState;
dwParam2 = (DWORD) DevStateData1;
dwParam3 = (DWORD) DevStateData2;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—Unused.
- *dwMsg*—The value LINE_LINEDEVSTATE. Specifies the callback instance supplied when opening the line.
- *dwParam1*—Specifies the line device status item that has changed. This parameter uses 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 are numbers in the range one to *dwNumRingModes*, where dwNumRingModes is a line device capability.
- *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.

Usage Notes

- The sending of this message can be controlled through **TSPI_lineSetStatusMessages**. The service provider must send LINE_LINEDEVSTATE messages for at least the set of status changes selected through that procedure. The service provider can send more than this set, however, it should try to limit its messages to this set for performance reasons. By default all status reporting is disabled.
- At the TSPI level, the service provider does not report state changes when the line is opened and closed, since there is only ever one Open outstanding for the device.
- For backward compatibility, older service providers would not be expected to generate these values. If they do, TAPI treats them the same as if the service provider were using API version 1.4 or later (as described earlier).

LINE_NEWCALL

The TSPI **LINE_NEWCALL** message is sent to the LINEEVENT callback function whenever a new call that TAPI has not originated arrives on a line that TAPI has open. This must be the first message sent regarding that call. TAPI writes the *htCall* opaque handle to the location passed by the service provider as *dwParam2*. This gives the service provider the *htCall* value to be used in subsequent messages.

Message Detail

```
LINE_NEWCALL
```

```
htLine = (HTAPILINE) hLineDevice;
htCall = (HTAPICALL) 0;
dwMsg = (DWORD) LINE_NEWCALL;
dwParam1 = (DWORD)(HDRVCALL) hdCall;
dwParam2 = (DWORD)(LPHTAPICALL) &htCall;
dwParam3 = (DWORD) 0;
```

Parameters

- *htLine*—The TAPI opaque object handle to the line device.
- *htCall*—Unused.
- *dwMsg*—The value LINE_NEWCALL.
- *dwParam1*—The service provider's opaque handle for the call, of type HDRVCALL. TAPI passes this value as the *hdCall* parameter to identify the call in subsequent procedures it invokes to operate on the call.
- *dwParam2*—A pointer of type LPHTAPICALL pointing to a HTAPICALL. TAPI writes the TAPI opaque handle for the call to the indicated location. The service provider must save this value and pass it as the *htCall* parameter to identify the call in subsequent events it reports for the call. This parameter can also acquire a value of NULL (see the following Remarks section).
- *dwParam3*—Unused.

Usage Notes

- The service provider should send the LINE_CALLSTATE message as the next message for this call. The LINE_NEWCALL event is unusual in that it also passes a value back to the service provider.
- This function reports any new calls that originate in the service provider (inbound, outbound, initiated at the phone, and so on) for which TAPI and the service provider have not yet exchanged opaque handles. The handles are exchanged so that TAPI and the service provider can subsequently make requests and report events involving the call. Because these new calls are not necessarily inbound, the calls can initially be in *any* state, not necessarily the *offering* state.

  If the service provider starts and discovers that one or more calls are already active on the line, it informs TAPI of them with LINE_NEWCALL messages followed by LINE_CALLSTATE messages indicating the current state. A new outgoing call, initiated on the phone by the user, would be reported with a LINE_NEWCALL message, and the initial LINE_CALLSTATE message would indicate that the call was in DIALTONE state (and then continuing on from there).
- If the service provider passes a large number of calls to TAPI in a very short amount of time (during the same interrupt cycle), TAPI can become backlogged in processing those calls. When this happens, TAPI signals to the service provider to wait a short time before sending any more calls. It signals this by writing a value of NULL, instead of a valid HTAPICALL, into the location pointed to by the *dwParam2* parameter of LINE_NEWCALL. This indicates that the attempt to process the newly offered call handle was not successful, most likely due to a temporary inability to allocate memory.

  The service provider can respond by dropping the call or by resending the LINE_NEWCALL message after a scheduling delay (during which time the service provider should yield the processor to allow TAPI to process other pending actions). In any case, no further messages regarding the new call can be passed to TAPI until the handle exchange succeeds. When the location pointed to by *dwParam2* acquires a non-NULL value, the service provider knows that this value is a valid HTAPICALL handle to the call.
- There is no directly corresponding message at the TAPI level. This message is used at the TSPI level to uniquely and unambiguously introduce a new incoming call to TAPI and retrieve the TAPI opaque identifier for the call.

LINE_SENDDIALOGINSTANCEDATA

The TSPI **LINE_SENDDIALOGINSTANCEDATA** message causes TAPI to call the TUISPI_providerGenericDialogData function in the UI DLL associated with htDlgInst, passing it the parameter block pointed to by *lpParams*, of length dwSize.

Message Detail

```
LINE_SENDDIALOGINSTANCEDATA
htLine = (DWORD) htDlgInst;
htCall = (DWORD) 0;
dwMsg = LINE_SENDDIALOGINSTANCEDATA;
dwParam1 = (DWORD) lpParams;
dwParam2 = (DWORD) dwSize;
dwParam3 = (DWORD) 0;
```

Parameters

- *htDlgInst*—The HTAPIDIALOGINSTANCE that was returned to the service provider when the dialog box instance was created using LINE_CREATEDIALOGINSTANCE.
- *lpParams*—Pointer to a provider-specific parameter block that is conveyed to the UI DLL TUISPI_providerGenericDialogData function, the size of which is specified by dwSize. If this parameter is set to NULL, this is a request to close the dialog box immediately and clean up (the UI DLL should not invoke TUISPIDLLCALLBACK during this cleanup).
- *dwSize*—The size in bytes of the parameter block to be conveyed to the UI DLL.

Glossary

API—Application Programming Interface

CME—Cisco CallManager Express.

SRST—Survivable Remote Site Telephony.

TAPI—Telephony Application Programmer's Interface

TSP—TAPI Service Provider

TSPI—Telephony Service Provider Interface

Note For a list of other internetworking terms, see *Internetworking Terms and Acronyms* document that is available on the Documentation CD-ROM and on the Cisco Connection Online (CCO) at the following URL: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ita/index.htm.