IP Office 4.0

TAPI Link Developer’s Guide

15-601035 Issue 11 (6 December 2006)

Notice
While reasonable efforts were made to ensure that the information in this document was complete and accurate at the time of printing, Avaya Inc. can assume no liability for any errors. Changes and corrections to the information in this document may be incorporated in future releases.

Documentation Disclaimer
Avaya Inc. is not responsible for any modifications, additions, or deletions to the original published version of this documentation unless such modifications, additions, or deletions were performed by Avaya.

Link Disclaimer
Avaya Inc. is not responsible for the contents or reliability of any linked Web sites referenced elsewhere within this Documentation, and Avaya does not necessarily endorse the products, services, or information described or offered within them. We cannot guarantee that these links will work all of the time and we have no control over the availability of the linked pages.

License
USE OR INSTALLATION OF THE PRODUCT INDICATES THE END USER'S ACCEPTANCE OF THE TERMS SET FORTH HEREIN AND THE GENERAL LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE AT http://support.avaya.com/LicenseInfo/ ("GENERAL LICENSE TERMS"). IF YOU DO NOT WISH TO BE BOUND BY THESE TERMS, YOU MUST RETURN THE PRODUCT(S) TO THE POINT OF PURCHASE WITHIN TEN (10) DAYS OF DELIVERY FOR A REFUND OR CREDIT.

Avaya grants End User a license within the scope of the license types described below. The applicable number of licenses and units of capacity for which the license is granted will be one (1), unless a different number of licenses or units of capacity is specified in the Documentation or other materials available to End User. "Designated Processor" means a single stand-alone computing device. "Server" means a Designated Processor that hosts a software application to be accessed by multiple users. "Software" means the computer programs in object code, originally licensed by Avaya and ultimately utilized by End User, whether as stand-alone Products or pre-installed on Hardware. "Hardware" means the standard hardware Products, originally sold by Avaya and ultimately utilized by End User.

License Type(s): Designated System(s) License (DS).
End User may install and use each copy of the Software on only one Designated Processor, unless a different number of Designated Processors is indicated in the Documentation or other materials available to End User. Avaya may require the Designated Processor(s) to be identified by type, serial number, feature key, location or other specific designation, or to be provided by End User to Avaya through electronic means established by Avaya specifically for this purpose.

Copyright
Except where expressly stated otherwise, the Product is protected by copyright and other laws respecting proprietary rights. Unauthorized reproduction, transfer, and or use can be a criminal, as well as a civil, offense under the applicable law.

Third-Party Components
Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements ("Third Party Components"), which may contain terms that expand or limit rights to use certain portions of the Product ("Third Party Terms"). Information identifying Third Party Components and the Third Party Terms that apply to them is available on Avaya's web site at: http://support.avaya.com/ThirdPartyLicense/

Avaya Fraud Intervention
If you suspect that you are being victimized by toll fraud and you need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. Suspected security vulnerabilities with Avaya Products should be reported to Avaya by sending mail to: securityalerts@avaya.com.

For additional support telephone numbers, see the Avaya Support web site (http://www.avaya.com/support).

Trademarks
Avaya and the Avaya logo are registered trademarks of Avaya Inc. in the United States of America and other jurisdictions. Unless otherwise provided in this document, marks identified by "®," "™" and "SM" are registered marks, trademarks and service marks, respectively, of Avaya Inc. All other trademarks are the property of their respective owners.

Documentation information
For the most current versions of documentation, go to the Avaya Support web site (http://www.avaya.com/support) or the IP Office Knowledge Base (http://marketingtools.avaya.com/knowledgebase/).

Avaya Support
Avaya provides indirect and direct services for customer support, report problems or to ask questions about your product. These services are subject to your support agreement. Contact your local reseller / distributor for indirect support . Contact Avaya Global Services (AGS) for direct support. For additional information on support, see the Avaya Web site: http://www.avaya.com/support.

Table Of Contents

IP Office TAPI Link

IP Office TAPI Link

Overview

The IP Office CTI Link is available in Lite and Pro versions, which provide run-time interfaces for applications to use. The Software Development Kit (SDK) provides documentation on both Lite and Pro interfaces for software developers.

Both the Lite and Pro offerings are the same program. The additional functionality provided by IP Office CTI Link Pro is enabled when the CTI Link Pro licence key is installed. Refer to the IP Office CTI Link Installation Manual for details.

This document provides information to assist a developer to implement an application that uses the IP Office TAPI Service Provider. It also assumes the developer is already familiar with TAPI. It is recommended that the reader of this document has access to the Microsoft Developer Network (MSDN) Library, which provides a complete TAPI reference.

IP Office TAPI Driver

The architecture of Windows allows developers to implement applications using standard Application Programming Interfaces (API) regardless of telephony equipment being used. Telephony equipment manufacturers provide telephony drivers, called Telephony Service Providers (TSP), that are installed on Windows. These TSPs provide the link between TAPI and the telephony equipment.

The TAPI driver for IP Office supports all TAPI versions from 2.0 to 3.0.

Disclaimer

Please note that although Avaya intend that releases of the IP Office TAPI Driver will provide backwards compatibility with earlier versions of the IP Office TAPI Driver, in terms of the feature set provided, Avaya cannot guarantee that the behaviour of IP Office will remain unchanged. Due to improvements in IP Office, the precise sequence, timing and content of TAPI events are likely to change. It is recommended that developers use an event driven programming model to make their applications resilient to such changes.

References

The following are recommended reading:

- MSDN/Platform SDK
- Windows Telephony Programming (TAPI 1.x and 2.x)
- CTI Link Installation Manual

Structure of this Document

The remainder of this document consists of the following sections:

- TAPI 2 Reference
    - Alphabetical reference of TAPI functions
    - Alphabetical reference of TAPI structures
    - Alphabetical reference of TAPI events
- TAPI 3 Reference
- TAPI 3 Enumeration Types
- Media Service Provider
- Communication Loss and Recovery

Installing the TAPILink and Wave Drivers

The IP Office TAPI Service Provider and Wave Driver are installed from the IP Office User CD.

Refer to the CTI Link Installation Manual for details.

Installing the CTI TAPI Linkpro License and Wave Licenses

You do not need a license in order to use the TAPI driver, but the license provides the following additional functionality:

- Third Party mode
- ACD Queue monitoring
- lineDevSpecific function enabled

To use the Wave functionality you need to install a *Wave User's Licence* for each Wave user, in addition to the CTI Link Pro license.

Configuring the TAPI Driver

The IP Office TAPI Service Provider can operate in Single User mode or Third Party mode. A license must be purchased to enable the Third Party mode. Note that the unlicensed (Lite) version will not prevent you from selecting this option but it will not work.

Single User mode means that the TAPI application can control and/or monitor a single telephony device. Third Party mode means that the TAPI application can control and/or monitor all telephony devices on a particular IP Office unit.

Single User Mode

Enter the IP address of the IP Office unit in the box labelled **Switch IP Address**. Select the **Single User** option. Enter the user name and password for the extension that is to be monitored and/or controlled by TAPI. Normally, the user name will be the name of a person associated with a physical telephone extension.

Third Party Mode

Enter the IP address of the IP Office unit in the box labelled **Switch IP Address**. Select the **Third Party** option. Enter the password for the switch. This is the same password that is entered in Manager when loading the configuration of the switch.

By default, Third Party mode will provide a TAPI line for every physical extension attached to the IP Office. The check boxes associated with Third Party mode enable additional entities to be monitored and/or controlled by TAPI.

ACD Queues

The IP Office can be configured to queue incoming calls that are being presented to a group of internal users. For example, if your IP Office was configured with a group of call center agents, you would want to queue an incoming call until an agent becomes available to take the call.

Checking the ACD Queues check box provides lines to monitor and/or control the queue of calls against a group.

WAV Users

If a user has a user name that begins with "TAPI:" it is a WAV user. The IP Office switch will attempt to stream audio to WAV users when they are involved in calls.

This audio streaming requires the IP Office wave driver to be installed on the PC and requires a wave driver licence instance per user. If the wave driver is not installed, you may still have the WAV Users tick box checked and will still receive WAV user events without the need for a licence.

During use the TAPI WAV audio stream uses an IP Office data channel taken from the same pool of data channels as used for voicemail ports. The maximum number of data channels available for simultaneous voicemail and TAPI WAV calls depends on the IP Office control unit type; IP401 = 2, IP403 and Small Office Edition = 10, IP406 = 20, IP412 = 30.

Configuring your IP Office for TAPI

This section describes the configuration of the IP Office using the Manager application. If your application monitors telephones but does not control them, then there is no configuration necessary.

There are two ways in which you can use TAPI with IP Office:

- If your application controls telephones, you should configure all users that will be controlled as an **off-hook station**. This will cause the user's phone to return to the idle state when a call is hung up using TAPI. Without this option set, the phone will remain in a disconnected state until the phone is hung up manually. The off-hook station check box can be found on the Telephony tab of the User's setting in Manager.
- If you require a special user that will handle media streaming (such as an auto attendant), create a new user with a name that begins with "TAPI:". This will be a WAV user. The user's number should be in a range that does not conflict with any existing phone numbers or groups.

Overview

It is advisable to close all TAPI applications before resetting the switch. This allows the Telephony Service Provider (TSP) to gracefully close all open lines and ensures that the switch and all connected TSPs have a consistent state. In the event of an unexpected loss of communication (the switch is accidentally powered down or a network cable is accidentally unplugged), the TSP will detect that it is no longer connected to the switch.

During this time, any calls to TAPI functions that require the TSP to communicate with the switch, will be rejected. The time delay between communication being lost and the TSP detecting the loss depends on TCP settings on the host machine and internal timing in the TSP. The delay could be up to two minutes.

After the TSP has detected that it has lost communication with the switch, it will attempt to re-establish a connection. When the connection is re-established the service provider will usually be able to recover the open lines/addresses. This is the case even if the loss of communication was due to the switch rebooting.

The way in which the loss of communication appears to the TAPI application depends on the version of TAPI being used. This is described below.

TAPI 2

When the TSP loses its connection to IP Office, a LINEDEVSTATE_OUTOFSERVICE message will be sent on all open lines. When communication is re-established, a LINEDEVSTATE_INSERVICE message will be sent for each TAPI line recovered.

TAPI 3

When the TSP loses its connection to IP Office an ITAddressEvent is generated for each address that has registered for such events. These events will indicate that the addresses state has changed. The state will become AS_OUTOFSERVICE. When the TSP re-establishes its connection to IP Office no events are generated. However, once communication has been re-established, all open TAPI 3 Addresses will be recovered.

TAPI 2.x Reference

TAPI Functions

Overview

This section describes each of the TAPI 2.x functions supported by the IP Office TAPI driver. It describes any particular behaviour or limitations of the functions when used with IP Office.

- lineAddToConference
- lineAnswer
- lineBlindTransfer
- lineCompleteTransfer
- lineConfigDialog
- lineClose
- lineDeallocateCall
- lineDevSpecific
- lineDial
- lineDrop
- lineGenerateDigits
- lineGenerateTone
- lineGetAddressCaps
- lineGetAddressID
- lineGetAddressStatus
- lineGetAppPriority
- lineGetCallInfo
- lineGetCallStatus
- lineGetDevCaps
- lineGetID
- lineGetLineDevStatus
- lineHandoff
- lineHold
- lineInitializeEx
- lineMakeCall
- lineMonitorDigits
- lineMonitorTone
- lineNegotiateAPIVersion
- lineOpen
- linePark
- lineRedirect
- lineRemoveFromConference
- lineSetAppPriority
- lineSetAppSpecific
- lineSetCallPrivilege
- lineSetStatusMessages
- lineSetupTransfer
- lineShutdown
- lineSwapHold
- lineUnhold
- lineUnpark

lineAddToConference

Adds the call to the conference.

```
LONG
WINAPI
lineAddToConference(
HCALL hConfCall,
HCALL hConsultCall
);
```

lineAnswer

Answer a call that is being offered to the application.

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

Note

- "UserUserInfo" is not supported and will be ignored.

lineBlindTransfer

This function can be used to transfer an active call to a third party. The country code is ignored.

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

lineCompleteTransfer

This function can be used to complete a transfer or complete setting up a conference call. This function is supposed to return a call id to the conference but it always returns zero.

```
LONG
WINAPI
lineCompleteTransfer(
HCALL hCall,
HCALL hConsultCall,
LPHCALL lphConfCall,
DWORD dwTransferMode
);
```

lineConfigDialog

Displays the same TAPI Service Provider configuration dialog that appears in Control Panel/Phone and Modem options (or Telephony). Parameter lpszDeviceClass is ignored.

```
LONG

WINAPI

lineConfigDialog(

DWORD dwDeviceID,
HWND hwndOwner,
LPCSTR lpszDeviceClass
);
```

lineClose

Closes a line. Call this when you no longer want to make, receive or monitor calls on a line.

```
LONG

WINAPI

lineClose(

HLINE hLine
);
```

lineDeallocateCall

Deallocate resources associated with a call. This should be called once a call is in the idle state.

```
LONG

WINAPI

lineDeallocateCall(

HCALL hCall
);
```

lineDevSpecific

lineDevSpecific

The TSPI allows for extended functionality through the the lineDevSpecific function.

Note that this is only available in the licensed version of the TAPI driver.

TAPI's lineDevSpecific function takes a buffer and passes that buffer, unmodified through to the TSP where it is interpreted as device specific commands. The types of commands are described in the following paragraphs:

```
LONG

WINAPI

lineDevSpecific(

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

The Login Protocol

To log an ACD agent onto the line being monitored, set the first byte in the buffer to 8. The following bytes should be a character string, describing the extension that is logging on. So, consider the following buffer, used to log agent 218 onto the line on which we are calling lineDevSpecific: -

```
unsigned char buf[6];

int len = 6;

buf[0] = 8; // Constant that means Login

buf[1] = '2';

buf[2] = '1';

buf[3] = '8';

buf[4] = 0; // Don't forget the null terminator
```

Logging Off

Log off can be done by passing the following buffer to the DevSpecific function:

```
unsigned char buf[3];

int len = 3;

buf[0] = 9; // Constant that means Shortcode

buf[1] = 47; // Constant that means Log off

buf[2] = 0; // Don't forget the null terminator
```

Divert Destination

To set the target for diverted calls, send 9 in the first byte, 6 in the second and the following bytes should be a character string representing the divert destination extension. For example, to set the divert destination to extension 236, send the following buffer:

```
unsigned char buf[6];
int len = 6;
buf[0] = 9; // The first two bytes are devspecific constants
buf[1] = 6;
buf[2] = '2';
buf[3] = '3';
buf[4] = '6';
buf[5] = 0; // Don't forget the null terminator
```

Message Waiting Lamp

Some phones have lights that are lit when the user has voicemail messages waiting for them. The number of messages waiting can be controlled by a devspecific command. Zero messages will extinguish the lamp. One or more messages will light the lamp. Send the following buffer to lineDevSpecific:

```
unsigned char buf[21];
int len = 21;
buf[0] = 9; // Shortcode
buf[1] = 73; // Set MWL
sprintf(&(buffer[2]), ";Mailbox Msgs=%d", num);
// Where num is the number of messages
```

Note

- The IP Office server or other IP Office applications may also control the message waiting lamp.

Forward (Divert) Settings

The following constants will help with switching divert features on and off:

```
const unsigned char ForwardAllOn = 0;
const unsigned char ForwardAllOff = 1;
const unsigned char ForwardBusyOn = 2;
const unsigned char ForwardBusyOff = 3;
const unsigned char ForwardNoAnswerOn = 4;
const unsigned char ForwardNoAnswerOff = 5;
const unsigned char DoNotDisturbOn = 7;
const unsigned char DoNotDisturbOff = 8;
```

A buffer that uses any of these constants should be three bytes in length and should begin with a 9. For example, the following code will switch the line to 'Do Not Disturb':

```
unsigned char buf[3];
int len = 3;
buf[0] = 9;
buf[1] = DoNotDisturbOn;
buf[2] = 0;
```

Group Enable and Disable

Send the following buffer to enable the user's membership in the group with extension *groupnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9;
buf[1] = 76;
sprintf((char*)&buf[2], "%d", groupnum);
```

Send the following buffer to disable the user's membership in the group with extension *groupnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9;
buf[1] = 77;
sprintf((char*)&buf[2], "%d", groupnum);
```

In both cases (disabling and enabling group membership), you may elect to disable or enable *all* group membership by omitting the group number and placing a zero in its place (ie. buf[2] = 0).

Note

- You can only enable and disable a user from groups that the user is a member of, as configured in Manager.

Intrude

Send the following buffer to intrude upon another callers call. The call party to be intruded upon is identified by the integer *extnnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9; // Shortcode
buf[1] = 83; // Intrude
sprintf((char*)&buf[2], "%d", extnnum);
```

Listen

Send the following buffer to listen to another callers call. The call party to be listened to is identified by the integer *extnnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9; // Shortcode
buf[1] = 100; // Listen
sprintf((char*)&buf[2], "%d", extnnum);
```

lineDial

This function is used to dial a number on an existing call. It can be used as part of a supervised transfer (see lineSetupTransfer). Country code is ignored.

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

lineDrop

Hangs up a call. UserUserInfo is not supported and will be ignored.

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

lineGenerateDigits

Call this function to generate DTMF digits on the call. The user does not need to be a WAV user and the wave driver does not need to be involved in the call. A LINE_GENERATE message will be sent to the application when the generation is finished. The only dwDigitMode supported is LINEDIGITMODE_DTMF.

```
LONG

WINAPI

lineGenerateDigits(

HCALL hCall,
DWORD dwDigitMode,
LPCSTR lpszDigits,
DWORD dwDuration
);
```

lineGenerateTone

This function can be used to generate a beep on the line. The line must be a WAV user and the wave driver must be involved in the call. The only supported value for dwToneMode is LINETONEMODE_BEEP. As we do not support custom tones, dwNumTones should be zero.

```
LONG

WINAPI

lineGenerateTone(

HCALL hCall,
DWORD dwToneMode,
DWORD dwDuration,
DWORD dwNumTones,
LPLINEGENERATETONE const lpTones
);
```

lineGetAddressCaps

Retrieves the telephony capabilities of a particular address for a particular line. The capabilities are returned in the LINEADDRESSCAPS structure. See LINEADDRESSCAPS in the TAPI structures section for details.

IP Office lines always have a single address.

```
LONG

WINAPI

lineGetAddressCaps(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAddressID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
LPLINEADDRESSCAPS lpAddressCaps
);
```

lineGetAddressID

This function is used to map a phone number (address) assigned to a line device back to its *dwAddressID* in the range zero to the number of addresses minus one returned in the line's device capabilities (LINEDEVCAPS). Given that dwNumAddresses in LINEDEVCAPS is 1, this function will always return 0 in the DWORD pointed to by lpdwAddressID.

```
LONG

WINAPI

lineGetAddressID(

HLINE hLine,
LPDWORD lpdwAddressID,
DWORD dwAddressMode,
LPCSTR lpsAddress,
DWORD dwSize
);
```

lineGetAddressStatus

This function allows an application to query the specified address for its current status. See LINEADDRESSSTATUS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetAddressStatus(

HLINE hLine,
DWORD dwAddressID,
LPLINEADDRESSSTATUS lpAddressStatus
);
```

lineGetAppPriority

Retrieve your applications priority.

```
LONG

WINAPI

lineGetAppPriority(

LPCSTR lpszAppFilename,
DWORD dwMediaMode,
LPLINEEXTENSIONID lpExtensionID,
DWORD dwRequestMode,
LPVARSTRING lpExtensionName,
LPDWORD lpdwPriority
);
```

lineGetCallInfo

Obtain fixed information about the specified call. See LINECALLINFO structure for details.

```
LONG

WINAPI

lineGetCallInfo(

HCALL hCall,
LPLINECALLINFO lpCallInfo
);
```

lineGetCallStatus

This function retrieves a LINECALLSTATUS structure relating to an existing call. See LINECALLSTATUS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetCallStatus(

HCALL hCall,
LPLINECALLSTATUS lpCallStatus
);
```

lineGetDevCaps

Call this function to retrieve the LINEDEVCAPS structure. See LINEDEVCAPS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetDevCaps(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
LPLINEDEVCAPS lpLineDevCaps
);
```

lineGetID

Get the ID for a line when dwSelect is LINECALLSELECT_LINE.

```
LONG

WINAPI

lineGetID(

HLINE hLine,
DWORD dwAddressID,
HCALL hCall,
DWORD dwSelect,
LPVARSTRING lpDeviceID,
LPCSTR lpszDeviceClass
);
```

lineGetLineDevStatus

The lineGetLineDevStatus returns a device specific buffer. The devspecific buffer contains the following information:

```
LONG

WINAPI

lineGetLineDevStatus(

HLINE hLine,
LPLINEDEVSTATUS lpLineDevStatus
);
```

Byte	Contains	Comment
0..n	Phone Extension	This is the line number being monitored, as a character string (eg. "217")
n+1	0	Null terminator for the string above.
n+2	Forward on busy	1 if the phone is set to forward on busy, 0 otherwise.
n+3	Forward on no answer	1 if the phone is set to forward on no answer, 0 otherwise.
n+4	Forward unconditional	1 if the phone is set to forward all.
n+5	Forward hunt group flag	1 if the phone is set to forward hunt group calls.
n+6	Do Not Disturb	1 if the phone is set to DND
n+7	Outgoing call bar flag	1 if the phone is barred from making external calls
n+8	Call waiting on flag	1 if call waiting is enabled for this phone
n+9	Voicemail on flag	1 if voicemail is enabled for this phone
n+10	Voicemail ring-back flag	1 if voicemail ringback is enabled for this phone
n+11	Number of read voicemail messages	The number of read messages.
n+12	Number of unread voicemail messages	The number of voicemail messages waiting for the user.
n+13	Outside call sequence number	Type of ring for external calls.
n+14	Inside call sequence number	Type of ring for internal calls.
n+15	Ring back sequence number	Type of ring for ringback calls.
n+16	No answer timeout period	Number of seconds the phone will ring before following the *no answer* action, e.g. forward on no answer, divert to voicemail.
n+17	Wrap up time period	Number of seconds the phone will remain unable to accept calls following a call.
n+18	Can intrude flag	1 if this phone can intrude upon calls.
n+19	Cannot be intruded upon flag	1 if this phone cannot be intruded upon.

n+20	X directory flag	1 if this user does not appear in the internal directory.
n+21	Force login flag	in the logged-out state on power up and therefore a user must log in.
n+22	Forced account code flag	1 if this phone is forced to provide a valid account code when making external calls.
n+23	Login code flag	1 if this user has a login code configured.
n+24	System phone flag	1 if this is a system phone
n+25	Absent message id	The id of the absent message
n+26	Absent message set flag	1 if the absent message with the id in the previous field is displayed on the phone.
n+27	Voicemail email mode	1 if voicemail email mode is enabled
n+28..m	Extn	The user extension, which may be different to the phone extension.
m+1	0	Null terminator for Extn string above.
m+2..p	locale	The locale of the user.
p+1	0	Null terminator for locale
p+2..q	Forward destination	The number that this phone is set to divert to
q+1	0	Null terminator for destination above.
q+2..r	Follow me number	All calls are redirected to this number.
r+1	0	
r+2..s	Absent text	The absent text defined for this phone.
s+1	0	
s+2..t	Do not disturb exception list	A list of numbers that are permitted to ring the phone while it is in the Do Not Disturb state. Each number is a null-terminated string. The last number in the list is terminated by two nulls (field "t+1" represents the second of these two nulls). If the list is empty then the data will contain just a single null (represented by t+1).
t+1	0	
t+2..u	Forward on busy number	The number that calls will divert to when this phone is busy.
u+1	0	
u+2	User's priority	This priority will be associated with all calls made by this user.
u+3	Group membership	This byte contains the number of groups that the user is currently enabled in.
u+4	Groups out of time	Number of groups that the user is a member of that are currently outside their time profile
u+5	Disabled groups	Number of groups the user is currently disabled from
u+6	Groups out of service	Number of groups that the user is a member of that

		are currently out of service
u+7	Night service groups	Number of groups that the user is a member of that are currently on night service

lineHandoff

The lineHandoff function gives ownership of the specified call to another application.

```
LONG

WINAPI

lineHandoff(

HCALL hCall,
LPCSTR lpszFileName,
DWORD dwMediaMode
);
```

lineHold

This function holds an active call.

```
LONG

WINAPI

lineHold(

HCALL hCall
);
```

lineInitializeEx

This is the first TAPI function that should be called to initialise TAPI. The lpdwAPIVersion parameter should be set to at least 0x00020000.

```
LONG

WINAPI

lineInitializeEx(

LPHLINEAPP lphLineApp,
HINSTANCE hInstance,
LINECALLBACK lpfnCallback,
LPCSTR lpszFriendlyAppName,
LPDWORD lpdwNumDevs,
LPDWORD lpdwAPIVersion,
LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams
);
```

lineMakeCall

This function makes a call. See section on call parameters at the end of the TAPI functions chapter.

```
LONG

WINAPI

lineMakeCall(

HLINE hLine,
LPHCALL lphCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode,
LPLINECALLPARAMS const lpCallParams
);
```

lineMonitorDigits

Call this function to enable the detection of DTMF digits. This function only works when the IP Office wave driver is involved in the call and the user is a WAV user (see the "WAV users" section). Detection is done by analysing media samples in the WAV driver. When a DTMF tone is detected a LINE_MONITORDIGITS message is sent to the application. dwDigitModes can be LINEDIGITMODE_DTMF and/or LINEDIGITMODE_DTMFEND. Call lineMonitorDigits with a dwDigitMode of zero to cancel DTMF digit detection.

```
LONG

WINAPI

lineMonitorDigits(

HCALL hCall,
DWORD dwDigitModes
);
```

lineMonitorTone

This function, like the one above, requires that the wave driver be involved in the call. Furthermore, it can only be used to detect silence. The frequencies in the LINEMONITORTONE structure pointed to by lpToneList must all be zero. If silence is detected, a LINE_MONITORTONE message is sent to the application. Call lineMonitorTone with lpToneList set to NULL to cancel silence detection.

```
LONG

WINAPI

lineMonitorTone(

HCALL hCall,
LPLINEMONITORTONE const lpToneList,
DWORD dwNumEntries
);
```

lineNegotiateAPIVersion

This function should be called immediately after lineInitializeEx to ensure that correct TAPI notifications are sent to your application. It must be called for every line that your application uses.

```
LONG

WINAPI

lineNegotiateAPIVersion(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAPILowVersion,
DWORD dwAPIHighVersion,
LPDWORD lpdwAPIVersion,
LPLINEEXTENSIONID lpExtensionID
);
```

lineOpen

This function opens a line device.

dwMediaModes should be set to LINEMEDIAMODE_INTERACTIVEVOICE for ISDN / T1 and LINEMEDIAMODE_UNKNOWN for Analogue trunks. You can specify both to handle calls from both trunk types.

```
LONG

WINAPI

lineOpen(

HLINEAPP hLineApp,
DWORD dwDeviceID,
LPHLINE lphLine,
DWORD dwAPIVersion,
DWORD dwExtVersion,
DWORD dwCallbackInstance,
DWORD dwPrivileges,
DWORD dwMediaModes,
LPLINECALLPARAMS const lpCallParams
);
```

Note

- If an attempt is made to open a line that is associated with a Wave user and no there is no Wave User license installed in IP Office, lineOpen will return LINEERR_RESOURCEUNAVAIL. For an explanation of Wave Users, see WAV Users.

linePark

This function parks a call. Only park mode LINEPARKMODE_DIRECTED is supported. The park address may be any alphanumeric string, however, only numeric digits can be entered from a telephone, so you may want to restrict your park addresses to numeric strings. The four default park addresses that appear in Phone Manager and eConsole are 1, 2, 3 and 4. You should use these numbers if you want parked calls to be unparked using these applications using the default configuration.

```
LONG

WINAPI

linePark(

HCALL hCall,
DWORD dwParkMode,
LPCSTR lpszDirAddress,
LPVARSTRING lpNonDirAddress
);
```

lineRedirect

The lineRedirect function redirects the specified offering call to the specified destination address. Country code is ignored.

```
LONG

WINAPI

lineRedirect(

HCALL hCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode
);
```

lineRemoveFromConference

Removes the call from the conference.

```
LONG

WINAPI

lineRemoveFromConference(

HCALL hCall
);
```

lineSetAppPriority

Call this to indicate your applications priority.

```
LONG

WINAPI

lineSetAppPriority(

LPCSTR lpszAppFilename,
DWORD dwMediaMode,
LPLINEEXTENSIONID lpExtensionID,
DWORD dwRequestMode,
LPCSTR lpszExtensionName,
DWORD dwPriority
);
```

lineSetAppSpecific

This function enables an application to set the application-specific field of the specified call's call-information record.

```
LONG

WINAPI

lineSetAppSpecific(

HCALL hCall,
DWORD dwAppSpecific
);
```

lineSetCallPrivilege

Call this to change your applications ownership rights to a particular call.

```
LONG

WINAPI

lineSetCallPrivilege(

HCALL hCall,
DWORD dwCallPrivilege
);
```

lineSetStatusMessages

This function enables the application to state which notification messages it requires. Typically, dwLineStates is set to LINEDEVSTATE_ALL, and dwAddressStates is set to LINEADDRESSSTATE_ALL.

```
LONG

WINAPI

lineSetStatusMessages(

HLINE hLine,
DWORD dwLineStates,
DWORD dwAddressStates
);
```

lineSetupTransfer

This function is called to create a consultation call in order to perform a supervised transfer. The call that is to be transferred must exist already. The call may be either active or on hold when this function is called. If the call is active it will be put on hold by this function. Call lineDial to ring the party that is to be transferred to. Call lineCompleteTransfer to complete the transfer.

```
LONG

WINAPI

lineSetupTransfer(

HCALL hCall,
LPHCALL lphConsultCall,
LPLINECALLPARAMS const lpCallParams
);
```

lineShutdown

Finish using TAPI line functions. Normally called as your application closes down.

```
LONG

WINAPI

lineShutdown(

HLINEAPP hLineApp
);
```

lineSwapHold

This function puts the current active call on hold and retrieves the held call.

```
LONG

WINAPI

lineSwapHold(

HCALL hActiveCall,
HCALL hHeldCall
);
```

lineUnhold

This function retrieves a held call. If the line is ringing a third party or has an active call with a third party, when this function is called, then the ringing/active call will be dropped before the held call is retrieved.

```
LONG

WINAPI

lineUnhold(

HCALL hCall
);
```

lineUnpark

This function retrieves a parked call. dwAddressID should be 0 because IP Office lines only have one address. lpszDestAddress should be the same identifier that was used to park the call (see linePark).

```
LONG

WINAPI

lineUnpark(

HLINE hLine,
DWORD dwAddressID,
LPHCALL lphCall,
LPCSTR lpszDestAddress
);
```

TAPI Structures

LINEADDRESSCAPS

This structure is returned by the lineGetAddressCaps function. The following table indicates the values that are returned for lines that relate to the IP Office TAPI driver.

Note

- Not all members of this structure are listed. For full information on the LINEADDRESSCAPS, see the Microsoft TAPI documentation.

Member	Description / Value
dwLineDeviceID	The ID of the line to which this address relates.
dwDevSpecificSize	No extra information specific to the device is passed.
dwDevSpecificOffset	0
dwAddressSharing	LINEADDRESSSHARING_PRIVATE
dwAddressStates	0
dwCallInfoStates	Returns the possible call info states which are:- LINECALLINFOSTATE_CALLID LINECALLINFOSTATE_RELATEDCALLID LINECALLINFOSTATE_NUMOWNERINCR LINECALLINFOSTATE_NUMOWNEDECR LINECALLINFOSTATE_NUMMONITORS LINECALLINFOSTATE_CALLERID LINECALLINFOSTATE_CALLEDID LINECALLINFOSTATE_REDIRECTIONID LINECALLINFOSTATE_REDIRECTINGID LINECALLINFOSTATE_DISPLAY LINECALLINFOSTATE_MONITORMODES LINECALLINFOSTATE_CALLDATA
dwCallerIDFlags	Returns the possible caller ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwCalledIDFlags	Returns the possible called ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME

	LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwConnectedIDFlags	Returns the possible connected ID flags which are:- LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwRedirectionIDFlags	Returns the possible redirection ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwRedirectingIDFlags	Returns the possible redirecting ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwCallStates	Returns the possible call states which are:- LINECALLSTATE_IDLE (the call no longer exists) LINECALLSTATE_OFFERING (a new call has arrived) LINECALLSTATE_ACCEPTED (the call has been claimed by a application) LINECALLSTATE_DIALTONE (the caller hears dial tone) LINECALLSTATE_DIALING (the switch is receiving dialling information) LINECALLSTATE_RINGBACK (the caller hears ringing) LINECALLSTATE_BUSY (the caller hears the busy signal) LINECALLSTATE_CONNECTED (the caller has been connected end to end) LINECALLSTATE_PROCEEDING (dialling has completed but the call has not yet been connected) LINECALLSTATE_ONHOLD (the call is on hold) LINECALLSTATE_CONFERENCED (the call is on a conference) LINECALLSTATE_ONHOLDPENDCONF (the call

	is on hold before being conferenced) LINECALLSTATE_ONHOLDPENDTRANSFER (the call is on hold before being transferred) LINECALLSTATE_DISCONNECTED (The other end has dropped the call) LINECALLSTATE_UNKNOWN (the call state is unknown)
dwDialToneModes	Returns the possible dial tone mode of LINEDIALTONEMODE_UNAVAIL
dwBusyModes	Returns the possible busy modes of LINEBUSYMODE_UNAVAIL
dwSpecialInfo	Returns the possible special info of LINESPECIALINFO_UNAVAIL
dwDisconnectModes	Returns the possible disconnect modes which are:- LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_PICKUP LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_BADADDRESS LINEDISCONNECTMODE_UNREACHABLE LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_INCOMPATIBLE LINEDISCONNECTMODE_UNAVAIL LINEDISCONNECTMODE_NODIALTONE LINEDISCONNECTMODE_QOSUNAVAIL LINEDISCONNECTMODE_BLOCKED LINEDISCONNECTMODE_DONOTDISTURB
dwMaxNumActiveCalls	The maximum number of active calls: 1
dwMaxNumOnHoldCalls	The maximum number of calls on hold: 9
dwMaxNumOnHoldPendingCalls	The maximum number of calls on hold pending: 9
dwMaxNumConference	The maximum number of conference calls: 9
dwMaxNumTransConf	The maximum number of transferred conference calls: 9
dwAddrCapFlags	Returns the possible address cap flags which are: LINEADDRCAPFLAGS_FWDNUMRINGS LINEADDRCAPFLAGS_DIALED LINEADDRCAPFLAGS_TRANSFERHELD

	LINEADDRCAPFLAGS_TRANSFERMAKE LINEADDRCAPFLAGS_CONFERENCEHELD LINEADDRCAPFLAGS_CONFERENCEMAKE LINEADDRCAPFLAGS_FWDSTATUSVALID
dwCallFeatures	Returns the possible call features which are:- LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_ANSWER LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_HOLD LINECALLFEATURE_PARK LINECALLFEATURE_REDIRECT LINECALLFEATURE_REMOVEFROMCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_SWAPHOLD LINECALLFEATURE_UNHOLD LINECALLFEATURE_SETCALLDATA
dwRemoveFromConfCaps	Returns the possible remove from conference caps which is LINEREMOVEFROMCONF_ANY.
dwRemoveFromConfState	Returns the possible remove from conference state which is LINECALLSTATE_ONHOLD.
dwTransferModes	Returns the possible transfer modes which are: LINETRANSFERMODE_TRANSFER LINETRANSFERMODE_CONFERENCE
dwParkModes	Returns the possible park mode of LINEPARKMODE_DIRECTED.
dwForwardModes	Returns the possible forward modes which are:- LINE FORWARDMODE_UNCOND LINE FORWARDMODE_UNCONDEXTERNAL LINE FORWARDMODE_UNCONDSPECIFIC LINE FORWARDMODE_BUSY LINE FORWARDMODE_BUSYINTERNAL LINE FORWARDMODE_BUSYEXTERNAL LINE FORWARDMODE_BUSYSPECIFIC LINE FORWARDMODE_NOANSW LINE FORWARDMODE_NOANSWINTERNAL LINE FORWARDMODE_NOANSWEXTERNAL

	LINE FORWARDMODE_NOANSWSPECIFIC LINE FORWARDMODE_BUSYNA LINE FORWARDMODE_BUSYNAINTERNAL LINE FORWARDMODE_BUSYNAEXTERNAL LINE FORWARDMODE_BUSYNASPECIFIC
dwMaxForwardEntries	The maximum number of forwarded entries: 10
dwMaxSpecificEntries	The maximum number of specific entries: 10
dwMinFwdNumRings	The minimum forward number of rings: 1
dwMaxFwdNumRings	The maximum forward number of ring: 99
dwMaxCallCompletions	0
dwCallCompletionConds	0
dwCallCompletionModes	0
dwNumCompletionMessages	0
dwCompletionMsgTextEntrySize	0
dwCompletionMsgTextSize	0
dwCompletionMsgTextOffset	0
dwAddressFeatures	Return the possible address features which are:- LINEADDRFEATURE_FORWARD LINEADDRFEATURE_MAKECALL LINEADDRFEATURE_SETUPCONF LINEADDRFEATURE_UNPARK LINEADDRFEATURE_FORWARDFWD LINEADDRFEATURE_FORWARDDND
dwPredictiveAutoTransferStates	0
dwNumCallTreatments	0
dwCallTreatmentListSize	0
dwCallTreatmentListOffset	0
dwDeviceClassesSize	0
dwDeviceClassesOffset	0
dwMaxCallDataSize	The maximum call data size: 127
dwCallFeatures2	0
dwMaxNoAnswerTimeout	0

dwConnectedModes	0
dwOfferingModes	0
dwAvailableMediaModes	0

LINEADDRESSSTATUS

This structure is returned by lineGetAddressStatus.

Note

- Not all members of this structure are listed. For full information on the LINEADDRESSSTATUS, see the Microsoft TAPI documentation.

Member	Description / Value
dwNumInUse;	Always 1
dwNumActiveCalls;	Reflects the number of active calls.
dwNumOnHoldCalls;	Always 0
dwNumOnHoldPendCalls;	Always 0
dwAddressFeatures;	Indicates the capabilities which are:- LINEADDRFEATURE_MAKECALL LINEADDRFEATURE_SETUPCONF LINEADDRFEATURE_UNPARK
dwNumRingsNoAnswer;	5
dwForwardNumEntries;	Always 0
dwForwardSize; /	Always 0
dwForwardOffset;	Always 0
dwTerminalModesSize;	Always 0
dwTerminalModesOffset;	Always 0
dwDevSpecificSize;	Always 0

LINECALLINFO

This structure is returned by lineGetCallInfo.

Note

- Not all members of this structure are listed. For full information on the LINECALLINFO, see the Microsoft TAPI documentation.

Member	Description / Value
dwAddressID	Always 0
dwBearerMode	Returns the possible bearer mode which is:- LINEBEARERMODE_VOICE
dwRate	64000
dwMediaMode	Returns the possible media mode which is :- LINEMEDIAMODE_INTERACTIVEVOICE
dwAppSpecific	Set by application.
dwCallID	Call ID
dwCallParamFlags	Returns the possible call parameter flags which is:-LINECALLPARAMFLAGS_IDLE
dwCallStates	Returns the possible call states which are:- LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_DIALTONE LINECALLSTATE_DIALING LINECALLSTATE_RINGBACK LINECALLSTATE_BUSY LINECALLSTATE_CONNECTED LINECALLSTATE_PROCEEDING LINECALLSTATE_ONHOLD LINECALLSTATE_CONFERENCED LINECALLSTATE_ONHOLDPENDCONF LINECALLSTATE_ONHOLDPENDTRANSFER LINECALLSTATE_DISCONNECTED LINECALLSTATE_UNKNOWN
dwMonitorMediaModes	0
dwCountryCode	0
dwTrunk	0xFFFFFFFF
dwCommentSize	0
dwCommentOffset	0
dwUserUserInfoSize	0
dwUserUserInfoOffset	0

dwHighLevelCompSize	0
dwHighLevelCompOffset	0
dwLowLevelCompSize	0
dwLowLevelCompOffset	0
dwChargingInfoSize	0
dwChargingInfoOffset	0
dwTerminalModesSize	0
dwTerminalModesOffset	0
dwCallDataOffset	0
dwSendingFlowspecSize	0
dwSendingFlowspecOffset	0
dwReceivingFlowspecSize	0
dwReceivingFlowspecOffset	0
dwCallerIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwCalledIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwConnectedIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwRedirectionIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwRedirectingIDAddressType	0 – only valid for TAPI Version 3.0 and above

LINECALLPARAMS

The following parameters are recognized in the LINECALLPARAMS structure that can be passed to lineMakeCall and lineSetupTransfer:

Note

- Not all members of this structure are listed. For full information on the LINECALLPARAMS, see the Microsoft TAPI documentation.

Member	Description/Value
dwCallParamFlags	Set this to zero for normal use or enter LINEBEARERMODE_VOICE if you wish to conceal the caller line identifier on the call.
dwCalledPartyOffset	Can be used to set the called party identifier.
dwCallingPartyIDOffset	Can be used to set the calling party identifier.

LINECALLSTATUS

This structure is returned by the lineGetCallStatus function.

Note

- Not all members of this structure are listed. For full information on the LINECALLSTATUS, see the Microsoft TAPI documentation.

Member	Description/Value
dwCallState	Returns one of the following states: LINECALLSTATE_IDLE (The call no longer exists) LINECALLSTATE_OFFERING (a new call has arrived) LINECALLSTATE_ACCEPTED (the call has been claimed by an application) LINECALLSTATE_DIALTONE (the caller hears a dial tone) LINECALLSTATE_DIALING (the switch is receiving dialling information) LINECALLSTATE_RINGBACK (the caller hears ringing) LINECALLSTATE_BUSY (the caller hears the busy signal) LINECALLSTATE_CONNECTED (the call has been connected end to end) LINECALLSTATE_PROCEEDING (dialling has completed but the call has not yet been connected) LINECALLSTATE_ONHOLD (the call is on hold) LINECALLSTATE_CONFERENCED (the call is on a conference) LINECALLSTATE_ONHOLDPENDCONF (the call is on hold before being conferenced) LINECALLSTATE_ONHOLDPENDTRANSFER (the call is on hold before being transferred) LINECALLSTATE_DISCONNECTED (the other end has dropped the call) LINECALLSTATE_UNKNOWN (the call state is unknown)
dwCallStateMode	Always zero.
dwCallPrivilege	The applications privilege for this call.
dwCallFeatures	The call features available for the call state indicated by dwCallState. TAPI specifices all possible features, however, only those that appear in dwCallFeatures in the LINEADDRESSCAPS structure can be used.
dwDevSpecificSize	0
dwDevSpecificOffset	0
dwCallFeatures2	0
tStateEntryTime	Zeros

LINEDEVCAPS

This structure is returned by the lineGetDevCaps function. The comments below indicate the values that will be returned for lines that relate to the IP Office TAPI driver.

Note

- Not all members of this structure are listed. For full information on the LINEDEVCAPS, see the Microsoft TAPI documentation.

Member	Description / Value
dwProviderInfoSize	Indicates the Provider Name, i.e. the name of the TSP.
dwSwitchInfoSize	0
dwPermanentLineID	Unique identifier assigned by Windows.
dwLineNameSize	Indicates the Line Name.
dwStringFormat	Returns the string format which is:- STRINGFORMAT_ASCII
dwAddressModes	Returns the address mode which is:- LINEADDRESSMODE_ADDRESSID
dwNumAddresses	1
dwBearerModes	Returns the bearer modes which are: LINEBEARERMODE_VOICE LINEBEARERMODE_SPEECH
dwMaxRate	0
dwMediaModes	Returns the media mode which is: LINEMEDIAMODE_INTERACTIVEVOICE
dwGenerateToneModes	Returns the generate tone mode which is:- LINETONEMODE_BEEP
dwGenerateToneMaxNumFreq	0
dwMonitorToneMaxNumFreq	1
dwMonitorToneMaxNumEntries	1
dwGatherDigitsMinTimeout	0
dwGatherDigitsMaxTimeout	0
dwMedCtlDigitMaxListSize	0
dwMedCtlMediaMaxListSize	0
dwMedCtlToneMaxListSize	0
dwMedCtlCallStateMaxListSize	0
dwDevCapFlags	Returns the dev cap flags which are:- LINEDEVCAPFLAGS_CLOSEDROP LINEDEVCAPFLAGS_DIALBILLING

	LINEDEVCAPFLAGS_DIALQUIET LINEDEVCAPFLAGS_DIALDUALTONE
dwMaxNumActiveCalls	9
dwAnswerMode	Returns the answer mode which is:- LINEANSWERMODE_NONE
dwRingModes	1
dwLineStates	Returns the line state which is:- LINEDEVSTATE_RINGING LINEDEVSTATE_CONNECTED LINEDEVSTATE_DISCONNECTED LINEDEVSTATE_INSERVICE LINEDEVSTATE_OUTOFSERVICE LINEDEVSTATE_OPEN LINEDEVSTATE_CLOSE LINEDEVSTATE_REINIT LINEDEVSTATE_TRANSLATECHNGE LINEDEVSTATE_REMOVED
dwUUIAcceptSize	0
dwUUIAnswerSize	100
dwUUIMakeCallSize	100
dwUUIDropSize	100
dwUUISendUserUserInfoSize	100
dwUUICallInfoSize	User to User call information size: 100
dwNumTerminals	0
dwTerminalCapsSize	0
dwTerminalCapsOffset	0
dwTerminalTextEntrySize	0
dwTerminalTextSize	0
dwTerminalTextOffset	0
dwDevSpecificSize	0
dwDevSpecificOffset	0
dwLineFeatures	Returns the line feature which is: LINEFEATURE_MAKECALL
dwSettableDevStatus	0
dwDeviceClassesSize	tapi\line
PermanentLineGuide	Only relevant if using TAPI Version 2.2 or

	higher.
dwAddressTypes	Only relevant if using TAPI Version 3.0 or higher.
ProtocolGuide	Only relevant if using TAPI Version 3.0 or higher.
dwAvailableTracking	Only relevant if using TAPI Version 3.0 or higher.

TAPI Events (Messages)

LINE_APPNEWCALL

A new call has been created.

LINE_CALLINFO

Information has changed in the LINECALLINFO structure.

LINE_CALLSTATE

The state of the call has changed. See dwCallStates in the LINEADDRESSCAPS structure for the list of states supported.

LINE_LINEDEVSTATE

The line device state has changed. The second parameter could be any one of the following:

- LINEDEVSTATE_DEVSPECIFIC - Devspecific information has changed.
- LINEDEVSTATE_CONNECTED, LINEDEVSTATE_DISCONNECTED - The connected state of the line has changed.
- LINEDEVSTATE_OUTOFSERVICE - The TSP has lost communication with the switch. This line is now out of service.
- LINEDEVSTATE_INSERVICE - The TSP had lost connection to the switch but has now recovered and the line is back in service.
- LINEDEVSTATE_RINGING - The switch has detected that the caller's phone is ringing.

LINE_DEVSPECIFIC

Notifies the application about device-specific events occurring on a line, address, or call. This message prompts the application to call lineGetLineDevStatus and analyse the devspecific buffer for changes.

LINE_ADDRESSSTATE

The status of an address has changed on a line that is currently open by the application.

TAPI 3.0 Reference

TAPI

TAPI

The TAPI object is created by CoCreateInstance. All other TAPI 3.0 objects are created by TAPI 3.0 itself.

ITTAPI

The ITTAPI interface is the base interface for the TAPI object.

Initialize

This is the first TAPI function that should be called to initialise TAPI.

```
HRESULT
Initialize();
```

Shutdown

Shuts down a TAPI session. Normally called as your applications closes down.

```
HRESULT
Shutdown();
```

EnumerateAddresses

This method enumerates the addresses that are currently available.

```
HRESULT
EnumerateAddresses ( IEnumAddress **ppEnumAddress );
```

RegisterCallNotifications

Sets which new call notifications an application will receive. The application must call the method for each address, indicating media type or types it can handle and specifying the privileges it requests.

```
HRESULT RegisterCallNotifications(
ITAddress *pAddress,
VARIANT_BOOL fMonitor,
VARIANT_BOOL fOwner,
long lMediaTypes,
long lCallbackInstance,
long *plRegister
);
```

put_EventFilter

The **put_EventFilter** method sets the event filter mask

```
HRESULT
put_EventFilter ( long lFilterMask );
```

Address

Address

The Address object represents an entity that can make or receive calls.

ITAddress

The interface is the base interface for the Address object.

get_AddressName

Gets the displayable name of the address.

```
HRESULT
get_AddressName (BSTR *ppName );
```

get_DialableAddress

The **get_DialableAddress** method gets the **BSTR,** which can be used to connect to this address.

```
HRESULT
get_DialableAddress (
BSTR *pDialableAddress
);
```

get_ServiceProviderName

The **get_ServiceProviderName** method gets the name of the Telephony Service Provider (TSP) that supports this address: for example, Unimdm.tsp for the Unimodem service provider or H323.tsp for the H323 service provider.

```
HRESULT
get_ServiceProviderName (
BSTR *ppName
);
```

CreateCall

The **CreateCall** method creates a new Call object that can be used to make an outgoing call and returns a pointer to the object's **ITBasicCallControl** interface.

```
HRESULT
CreateCall (
BSTR *pDialableAddress,
Long 1AddressType,
Long 1MediaTypes,
ITBasicCallControl **ppCall
);
```

IEnumAddress

Provides COM-standard enumeration methods for the ITAddress interface.

Next

The **Next** method gets the next specified number of elements in the enumeration sequence.

```
HRESULT
Next(
ULONG celt,
ITAddress **ppElements,
ULONG *pceltFetched
);
```

ITMediaSupport

The ITMediaSupport interface provides methods that allow an application to discover the media support capabilities for an Address Object that exposes this interface.

get_MediaTypes

The get_MediaTypes method gets the media type or types supported on the current address.

```
HRESULT
get_MediaTypes (
long *plMediaTypes
);
```

Terminal

Terminal object represents the source or sink of a media stream associated with a call or communications session.

Call

Call

The Call object represents an address's connection between the local address and one or more other addresses.

ITCallInfo

The ITCallInfo interface gets and sets a variety of information concerning a Call object.

get_Address

The **get_Address** method gets a pointer to the **ITAddress** interface of the Address object.

```
HRESULT
get_Address (
ITAddress **ppAddress
);
```

get_CallState

The **get_CallState** method gets a pointer to the current call state, such as CS_IDLE.

```
HRESULT
get_CallState (
CALL_STATE *pCallState
);
```

get_CallInfoString

The **get_CallInfoString** method gets a call information items described by a string, such as the displayable address.

```
HRESULT
get_CallInfoString (
CALLINFO_STRING CallInfoString,
BSTR *ppCallInfoString
);
```

SetCallInfoBuffer

Either by accident or design, TAPI 3.0 (Windows 2000) only allows this function on a call that is in the IDLE state. This has been changed in TAPI 3.1 (Windows XP) which allows call data to be set on calls in the connected state by passing CIB_CALLDATABUFFER as the CallInfoBuffer parameter.

```
HRESULT
SetCallInfoBuffer (
CALLINFO_BUFFER CallInfoBuffer,
DWORD dwSize
BYTE* pCallInfoBuffer
);
```

ITBasicCallControl

The ITBasicCallControl interface is used by the application to connect, answer, and perform basic telephony operations on a call object.

Connect

The Connect method attempts to complete the connection of an outgoing call.

```
HRESULT
Connect(
VARIANT_BOOL fSync
);
```

Answer

The Answer method answers an incoming call. This method can succeed only if the call state is CS_OFFERING.

```
HRESULT
Answer();
```

Disconnect

The Disconnect method disconnects the call. The call state will transition to CS_DISCONNECTED after the method completes successfully.

```
HRESULT
Disconnect(
DISCONNECT_CODE code
);
```

Hold

The **Hold** method places or removes the call from the hold.

```
HRESULT
Hold(
VARIANT_BOOL fHold
);
```

SwapHold

The **SwapHold** method swaps the call (which is active) with the specified call on hold.

```
HRESULT
SwapHold(
ITBasicCallControl *pCall
);
```

ParkDirect

The **ParkDirect** method parks the call at a specified address.

```
HRESULT
ParkDirect(
BSTR pParkAddress
);
```

Unpark

The **Unpark** method gets the call from park.

```
HRESULT
Unpark();
```

BlindTransfer

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

```
HRESULT BlindTransfer(
BSTR pDestAddress
);
```

Transfer

The **Transfer** method transfers the current call to the destination address.

```
HRESULT Transfer(
ITBasicCallControl *pCall,
VARIANT_BOOL fSync
);
```

Finish

The **Finish** method is called on a consultation call to finish a conference or a transfer.

```
HRESULT Finish(
FINISH_MODE finishMode
);
```

Conference

The **Conference** method adds a consultation call to the conference in which the current call is a participant.

```
HRESULT Conference(
ITBasicCallControl *pCall,
VARIANT_BOOL fSync
);
```

RemoveFromConference

The **RemoveFromConference** method removes the call from a conference if it is involved in one.

```
HRESULT RemoveFromConference();
```

ITCallStateEvent

The ITCallStateEvent interface contains methods that retrieve the description of call state events.

get_Cause

The **get_Cause** method gets the cause associated with this event.

```
HRESULT
get_Cause (
CALL_STATE_EVENT_CAUSE *pCEC
);
```

get_State

The **get_State** method gets information on the new call state.

```
HRESULT
get_State (
CALL_STATE *pCallState
);
```

get_Call

The **get_Call** method gets a pointer to the call information interface for the call on which the event has occurred.

```
HRESULT
get_Call
ITCallInfo **ppCallInfo
);
```

ITCallNotificationEvent

The ITCallNotificationEvent interface contains methods that retrieve the description of call notification events.

get_Call

The **get_Call** method returns the **ITCallInfo** interface on which a call event has occurred.

```
HRESULT
get_Call
ITCallInfo **ppCall
);
```

ITCallInfoChangeEvent

The ITCallInfoChangeEvent interface contains methods that retrieve the description of call information change events.

get_Call

The **get_Call** method returns the **ITCallInfo** interface on which call information has changed.

```
HRESULT
get_Call
ITCallInfo **ppCall
);
```

Call Hub

The Call Hub object exposes methods that retrieve information concerning participants in a multi-party call. Call Hubs are not supported by IP Office. Call Hub Events may be received but should be ignored.

TAPI 3 Enumerated Types

CALL_STATE

The CALL_STATE enum is used by the ITCallInfo::get_CallState and ITCallStateEvent::get_State methods.

Member	Value	Description
CS_IDLE	0	The call has been created, but **Connect** has not been called yet. A call can never transition into the idle state.
CS_INPROGRESS	1	**Connect** has been called, and the service provider is working on making a connection. This state is valid only on outgoing calls. This message is optional, because a service provider may have a call transition directly to the connected state.
CS_CONNECTED	2	Call has been connected to the remote end and communication can take place.
CS_DISCONNECTED	3	Call has been disconnected. There are several causes for disconnection. See the table of valid call state transitions below.
CS_OFFERING	4	A new call has appeared, and is being offered to an application. If the application has owner privileges on the call, it can either call **Answer** or **Disconnect** while the call is in the offering state.
CS_HOLD	5	The call is in the hold state.
CS_QUEUED	6	The call is queued.

CALLINFO_STRING

The **CALLINFO_STRING** enum is used by **ITCallInfo** methods that set and get call information involving the use of strings.

Member	Value	Description
CIS_CALLERIDNAME	0	The name of the caller.
CIS_CALLERIDNUMBER	1	The number of the caller.
CIS_CALLEDIDNAME	2	The name of the called location.
CIS_CALLEDIDNUMBER	3	The number of the called location.
CIS_CONNECTEDIDNAME	4	The name of the connected location.
CIS_CONNECTEDIDNUMBER	5	The number of the connected location.
CIS_REDIRECTIONIDNAME	6	The name of the location to which a call has been redirected.
CIS_REDIRECTIONIDNUMBER	7	The number of the location to which a call has been redirected.
CIS_REDIRECTINGIDNAME	8	The name of the location that redirected the call.
CIS_REDIRECTINGIDNUMBER	9	The number of the location that redirected the call.
CIS_CALLEDPARTYFRIENDLYNAME	10	The called party friendly name.
CIS_COMMENT	11	A comment about the call provided by the application that originated the call.
CIS_DISPLAYABLEADDRESS	12	A displayable version of the called or calling address.
CIS_CALLINGPARTYID	13	The identifier of the calling party.

DISCONNECT_CODE

The DISCONNECT_CODE enum is used by the ITBasicCallControl::Disconnect method.

Member	Value	Description
DC_NORMAL	0	The call is being disconnected as part of the normal cycle of the call.
DC_NOANSWER	1	The call is being disconnected because it has not been answered. (For example, an application may set a certain amount of time for the user to answer the call. If the user does not answer, the application can call **Disconnect** with the NOANSWER code.)
DC_REJECTED	2	The user rejected the offered call.

CALL_STATE_EVENT_CAUSE

The CALL_STATE_EVENT_CAUSE enum is returned by the ITCallStateEvent::get_Cause method.

Member	Value	Description
CEC_NONE	0	No call event has occurred.
CEC_DISCONNECT_NORMAL	1	The call was disconnected as part of the normal life cycle of the call (that is, the call was over, so it was disconnected).
CEC_DISCONNECT_BUSY	2	An outgoing call failed to connect because the remote end was busy.
CEC_DISCONNECT_BADADDRESS	3	An outgoing call failed because the destination address was bad.
CEC_DISCONNECT_NOANSWER	4	An outgoing call failed because the remote end was not answered.
CEC_DISCONNECT_CANCELLED	5	An outgoing call failed because the caller disconnected.
CEC_DISCONNECT_REJECTED	6	The outgoing call was rejected by the remote end.
CEC_DISCONNECT_FAILED	7	The call failed to connect for some other reason.

The IP Office Media Service Provider

About the MSP

The IP Office Media Service Provider serves a dual purpose. It providers media streaming capability which allows a TAPI 3 application to send and receive voice data on calls that are present on specific types of users' lines. It also allows an application access to device specific functionality of the IP Office.

Using The MSP

The media service provider interfaces are documented in the MSDN libraries. The DevSpice sample on the SDK CD gives an example of how to use the MSP for media streaming and device specific functionality. The MSP is available to every TAPI address that can be viewed in your TAPI 3 application. Media streaming capabilities are only available to addresses that are specifically named as WAVE users. WAVE users are users with a name that begins with "TAPI:" (Such as "TAPI:201"). You may create as many WAV users as you wish, but each WAVE user will require a wave driver licence instance to enable media streaming to that user.

Using the Device Specific Interfaces

The device specific interfaces are implemented on the Address and Call objects of the MSP. TAPI 3.0 will delegate queries for interfaces it does not recognise to the MSP. If, therefore, you have a pointer to an ITAddress interface, you can call QueryInterface to retrieve a pointer to the ITDivert interface (for example). The following code from the DevSpice sample illustrates:

```
ITDivert* pDivert = NULL;

if( SUCCEEDED( gpAddress->QueryInterface( IID_ITDIVERT,

(void**)&pDivert)))

{

DWORD dwDivertSettings = 0;
if( FAILED( pDivert-> GetDivertSettings(
&dwDivertSettings)))

{
```

The interfaces available from the address object are:

- ITACDAgent
- ITDivert
- ITGroup

The interface available from the Call object is:

- ITPlay

Furthermore, the address object acts as a connection point container for IP Office Private events. The connection point interface is available in the interfaces.h file of the DevSpice sample and is called IPOfficePrivateEvent. Details of these interfaces are given below.

ITACDAgent

IsLoggedIn(void)	Returns S_TRUE if the user is logged in and S_FALSE if the user is logged out.
LogOut(void)	Logs the user off of this line. The user must have "force logon" set in Manager.
Login(BSTR extn)	Logs the user onto the given extension.
CallListen(BSTR extn)	Listens to the call present at the given extension. The user must have the Can Intrude privilege set in Manager.
Intrude(BSTR extn)	Conferences the current user in to the call present at the given extension. The user must have the Can Intrude privilege set in Manager.
SetAccountCode(BSTR extn)	Sets the account code for the current call.

ITGroup

This interface contains functions to take the user in and out of group, as well as to intercept calls that present themselves at other phones in the group.

PickupAny(void)	Equivalent to executing the CallPickupAny shortcode on the user's terminal. See Manager for details.
PickupGroup(void)	Equivalent to executing the CallPickupGroup shortcode on the user's terminal.
PickupExtn(BSTR extn)	Equivalent to executing the CallPickupExtn shortcode on the user's extension.
PickupMembers(BSTR extn)	Equivalent to executing the CallPickupMembers shortcode on the user's extension.
Enable(BSTR groupextn)	Enables the user's membership of the given group. If groupextn is an empty string, the user will be enabled in all groups that he/she is a member of.
Disable(BSTR groupextn)	Disables the user's membership of the given group. If groupextn is an empty string, the user will be disabled in all groups that he/she is a member of.

ITDivert

This interface contains functions for getting and setting the divert flags for the address.

GetDivertAllDestination(BSTR* pDestination)	Gets the current Divert All destination and returns the result in the pDestination value.
SetDivertAllDestination(BSTR dest)	
GetDivertSettings(DWORD* pdwDivertSets)	This function sets bits in the DWORD pointed to by pdwDivertSets to indicate which of the divert settings are currently active. The bits are defined by the IP_OFFICE_DIVERT_SETTINGS enum (described below).
SetForwardAll(VARIANT_BOOL bOn)	Toggles the ForwardAll setting for this user.
SetForwardBusy(VARIANT_BOOL bOn)	Toggles the ForwardBusy setting for this user.
SetForwardNoAnswer(VARIANT_BOOL bOn)	Toggles the ForwardNoAnswer setting for this user.
SetDoNotDisturb(VARIANT_BOOL bOn)	Toggles the DND setting for this user.

The IP_OFFICE_DIVERT_SETTINGS enum is defined as follows:

```
typedef enum
{
IPOFF_FWDALL = 0x01,
IPOFF_FWDBUSY = 0x02,
IPOFF_NOANSWER = 0x04,
IPOFF_DND = 0x08,
IPOFF_DESTINATION = 0x10
} IP_OFFICE_DIVERT_SETTINGS;
```

Therefore, getting a result of 14 (0xe) from GetDivertSettings implies that the user has ForwardBusy, ForwardNoAnswer and DoNotDisturb set. The IPOFF_DESTINATION value is not used by GetDivertSettings, only by the Fire_DivertSettingsChanged function on the IPOfficePrivateEvents interface.

ITPlay

The ITPlay interface is implemented on the MSP Call object. It allows for recording and playing of wave files.

StartPlay(BSTR FileName)	FileName should be the complete path to a wave file to play.
StopPlay()	
StartRecord(BSTR FileName)	FileName should be the complete path to a wave file to record to.
StopRecord()	

Playing and recording can be stopped and started at any time on the call. Recording will use only a single file per call though, and will append to the file if recording is stopped and restarted. It is not advisable to attempt to record and play at the same time. If this is a requirement, recording and playing can be done by selecting terminals onto the call that supply audio data from a file, or record audio data to a file. TAPI 3.1 introduces file-streaming terminals to make this easier.

IPOfficePrivateEvents

This is a connection point interface that the MSP uses to report events on. See the DevSpice sample on how to register for, and handle, private events.

OnUserLogin(void)	Fired when an agent (a user with 'force logon' set in Manager) logs on.
OnUserLogout(void)	Fired when an agent logs out.
OnDivertSettingsChanged(DWORD dwDivertSettings)	Fired when the user changes one of their divert setting flags (such as Do Not Disturb or Divert On Busy) or the Divert All destination. The bits in the dwDivertSettings variable are set using the IP_OFFICE_DIVERT_SETTINGS enum described above.
OnGroupChanged(DWORD dwGroupCount)	Fired when the user enables or disables group membership. The dwGroupCount value gives the number of groups that this user is an enabled member of.
OnVoiceMail(DWORD dwNumMessages)	This is fired when the number of voicemail messages that the user has waiting for them changes. The new value is given in the dwNumMessages parameter.

Using the Media Streaming Capabilities of the MSP

The IP Office MSP handles media streaming to any wave user. In order to do this, you must select terminals onto the streams that the MSP exposes for a call. Details on how to do this are given in the MSDN and an example is shown in the DevSpice sample. It should be noted that IP Office streams are bi-directional, and there is only a single stream per call. This means that both capture and render terminals are accepted on the same stream (but only one of each).

The MSP encapsulates the functionality of the IP Office wave driver. The IP Office wave driver must be installed on each machine that wishes to do media streaming to wave users. If you do not wish to do media streaming, and only wish to monitor wave users using TAPI, you must ensure that the wave driver is not installed, or you will consume wave licence instances for every wave user line you open.

Overview

The IP Office CTI Link is available in Lite and Pro versions, which provide run-time interfaces for applications to use. The Software Development Kit (SDK) provides documentation on both Lite and Pro interfaces for software developers.

Both the Lite and Pro offerings are the same program. The additional functionality provided by IP Office CTI Link Pro is enabled when the CTI Link Pro licence key is installed. Refer to the IP Office CTI Link Installation Manual for details.

This document provides information to assist a developer to implement an application that uses the IP Office TAPI Service Provider. It also assumes the developer is already familiar with TAPI. It is recommended that the reader of this document has access to the Microsoft Developer Network (MSDN) Library, which provides a complete TAPI reference.

IP Office TAPI Driver

The architecture of Windows allows developers to implement applications using standard Application Programming Interfaces (API) regardless of telephony equipment being used. Telephony equipment manufacturers provide telephony drivers, called Telephony Service Providers (TSP), that are installed on Windows. These TSPs provide the link between TAPI and the telephony equipment.

The TAPI driver for IP Office supports all TAPI versions from 2.0 to 3.0.

Disclaimer

Please note that although Avaya intend that releases of the IP Office TAPI Driver will provide backwards compatibility with earlier versions of the IP Office TAPI Driver, in terms of the feature set provided, Avaya cannot guarantee that the behaviour of IP Office will remain unchanged. Due to improvements in IP Office, the precise sequence, timing and content of TAPI events are likely to change. It is recommended that developers use an event driven programming model to make their applications resilient to such changes.

References

The following are recommended reading:

- MSDN/Platform SDK
- Windows Telephony Programming (TAPI 1.x and 2.x)
- CTI Link Installation Manual

Structure of this Document

The remainder of this document consists of the following sections:

- TAPI 2 Reference
    - Alphabetical reference of TAPI functions
    - Alphabetical reference of TAPI structures
    - Alphabetical reference of TAPI events
- TAPI 3 Reference
- TAPI 3 Enumeration Types
- Media Service Provider
- Communication Loss and Recovery

Installing the TAPILink and Wave Drivers

The IP Office TAPI Service Provider and Wave Driver are installed from the IP Office User CD.

Refer to the CTI Link Installation Manual for details.

Installing the CTI TAPI Linkpro License and Wave Licenses

You do not need a license in order to use the TAPI driver, but the license provides the following additional functionality:

- Third Party mode
- ACD Queue monitoring
- lineDevSpecific function enabled

To use the Wave functionality you need to install a *Wave User's Licence* for each Wave user, in addition to the CTI Link Pro license.

Configuring the TAPI Driver

The IP Office TAPI Service Provider can operate in Single User mode or Third Party mode. A license must be purchased to enable the Third Party mode. Note that the unlicensed (Lite) version will not prevent you from selecting this option but it will not work.

Single User mode means that the TAPI application can control and/or monitor a single telephony device. Third Party mode means that the TAPI application can control and/or monitor all telephony devices on a particular IP Office unit.

Single User Mode

Enter the IP address of the IP Office unit in the box labelled **Switch IP Address**. Select the **Single User** option. Enter the user name and password for the extension that is to be monitored and/or controlled by TAPI. Normally, the user name will be the name of a person associated with a physical telephone extension.

Third Party Mode

Enter the IP address of the IP Office unit in the box labelled **Switch IP Address**. Select the **Third Party** option. Enter the password for the switch. This is the same password that is entered in Manager when loading the configuration of the switch.

By default, Third Party mode will provide a TAPI line for every physical extension attached to the IP Office. The check boxes associated with Third Party mode enable additional entities to be monitored and/or controlled by TAPI.

ACD Queues

The IP Office can be configured to queue incoming calls that are being presented to a group of internal users. For example, if your IP Office was configured with a group of call center agents, you would want to queue an incoming call until an agent becomes available to take the call.

Checking the ACD Queues check box provides lines to monitor and/or control the queue of calls against a group.

WAV Users

If a user has a user name that begins with "TAPI:" it is a WAV user. The IP Office switch will attempt to stream audio to WAV users when they are involved in calls.

This audio streaming requires the IP Office wave driver to be installed on the PC and requires a wave driver licence instance per user. If the wave driver is not installed, you may still have the WAV Users tick box checked and will still receive WAV user events without the need for a licence.

During use the TAPI WAV audio stream uses an IP Office data channel taken from the same pool of data channels as used for voicemail ports. The maximum number of data channels available for simultaneous voicemail and TAPI WAV calls depends on the IP Office control unit type; IP401 = 2, IP403 and Small Office Edition = 10, IP406 = 20, IP412 = 30.

Configuring Your IP Office for TAPI

This section describes the configuration of the IP Office using the Manager application. If your application monitors telephones but does not control them, then there is no configuration necessary.

There are two ways in which you can use TAPI with IP Office:

- If your application controls telephones, you should configure all users that will be controlled as an **off-hook station**. This will cause the user's phone to return to the idle state when a call is hung up using TAPI. Without this option set, the phone will remain in a disconnected state until the phone is hung up manually. The off-hook station check box can be found on the Telephony tab of the User's setting in Manager.
- If you require a special user that will handle media streaming (such as an auto attendant), create a new user with a name that begins with "TAPI:". This will be a WAV user. The user's number should be in a range that does not conflict with any existing phone numbers or groups.

Overview

It is advisable to close all TAPI applications before resetting the switch. This allows the Telephony Service Provider (TSP) to gracefully close all open lines and ensures that the switch and all connected TSPs have a consistent state. In the event of an unexpected loss of communication (the switch is accidentally powered down or a network cable is accidentally unplugged), the TSP will detect that it is no longer connected to the switch.

During this time, any calls to TAPI functions that require the TSP to communicate with the switch, will be rejected. The time delay between communication being lost and the TSP detecting the loss depends on TCP settings on the host machine and internal timing in the TSP. The delay could be up to two minutes.

After the TSP has detected that it has lost communication with the switch, it will attempt to re-establish a connection. When the connection is re-established the service provider will usually be able to recover the open lines/addresses. This is the case even if the loss of communication was due to the switch rebooting.

The way in which the loss of communication appears to the TAPI application depends on the version of TAPI being used. This is described below.

TAPI 2

When the TSP loses its connection to IP Office, a LINEDEVSTATE_OUTOFSERVICE message will be sent on all open lines. When communication is re-established, a LINEDEVSTATE_INSERVICE message will be sent for each TAPI line recovered.

TAPI 3

When the TSP loses its connection to IP Office an ITAddressEvent is generated for each address that has registered for such events. These events will indicate that the addresses state has changed. The state will become AS_OUTOFSERVICE. When the TSP re-establishes its connection to IP Office no events are generated. However, once communication has been re-established, all open TAPI 3 Addresses will be recovered.

TAPI 2.x Reference

TAPI Functions

Overview

This section describes each of the TAPI 2.x functions supported by the IP Office TAPI driver. It describes any particular behaviour or limitations of the functions when used with IP Office.

- lineAddToConference
- lineAnswer
- lineBlindTransfer
- lineCompleteTransfer
- lineConfigDialog
- lineClose
- lineDeallocateCall
- lineDevSpecific
- lineDial
- lineDrop
- lineGenerateDigits
- lineGenerateTone
- lineGetAddressCaps
- lineGetAddressID
- lineGetAddressStatus
- lineGetAppPriority
- lineGetCallInfo
- lineGetCallStatus
- lineGetDevCaps
- lineGetID
- lineGetLineDevStatus
- lineHandoff
- lineHold
- lineInitializeEx
- lineMakeCall
- lineMonitorDigits
- lineMonitorTone
- lineNegotiateAPIVersion
- lineOpen
- linePark
- lineRedirect
- lineRemoveFromConference
- lineSetAppPriority
- lineSetAppSpecific
- lineSetCallPrivilege
- lineSetStatusMessages
- lineSetupTransfer
- lineShutdown
- lineSwapHold
- lineUnhold
- lineUnpark

lineAddToConference

Adds the call to the conference.

```
LONG
WINAPI
lineAddToConference(
HCALL hConfCall,
HCALL hConsultCall
);
```

lineAnswer

Answer a call that is being offered to the application.

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

Note

- "UserUserInfo" is not supported and will be ignored.

lineBlindTransfer

This function can be used to transfer an active call to a third party. The country code is ignored.

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

lineCompleteTransfer

This function can be used to complete a transfer or complete setting up a conference call. This function is supposed to return a call id to the conference but it always returns zero.

```
LONG
WINAPI
lineCompleteTransfer(
HCALL hCall,
HCALL hConsultCall,
LPHCALL lphConfCall,
DWORD dwTransferMode
);
```

lineConfigDialog

Displays the same TAPI Service Provider configuration dialog that appears in Control Panel/Phone and Modem options (or Telephony). Parameter lpszDeviceClass is ignored.

```
LONG

WINAPI

lineConfigDialog(

DWORD dwDeviceID,
HWND hwndOwner,
LPCSTR lpszDeviceClass
);
```

lineClose

Closes a line. Call this when you no longer want to make, receive or monitor calls on a line.

```
LONG

WINAPI

lineClose(

HLINE hLine
);
```

lineDeallocateCall

Deallocate resources associated with a call. This should be called once a call is in the idle state.

```
LONG

WINAPI

lineDeallocateCall(

HCALL hCall
);
```

lineDevSpecific

lineDevSpecific

The TSPI allows for extended functionality through the the lineDevSpecific function.

Note that this is only available in the licensed version of the TAPI driver.

TAPI's lineDevSpecific function takes a buffer and passes that buffer, unmodified through to the TSP where it is interpreted as device specific commands. The types of commands are described in the following paragraphs:

```
LONG

WINAPI

lineDevSpecific(

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

The Login Protocol

To log an ACD agent onto the line being monitored, set the first byte in the buffer to 8. The following bytes should be a character string, describing the extension that is logging on. So, consider the following buffer, used to log agent 218 onto the line on which we are calling lineDevSpecific: -

```
unsigned char buf[6];

int len = 6;

buf[0] = 8; // Constant that means Login

buf[1] = '2';

buf[2] = '1';

buf[3] = '8';

buf[4] = 0; // Don't forget the null terminator
```

Logging Off

Log off can be done by passing the following buffer to the DevSpecific function:

```
unsigned char buf[3];

int len = 3;

buf[0] = 9; // Constant that means Shortcode

buf[1] = 47; // Constant that means Log off

buf[2] = 0; // Don't forget the null terminator
```

Divert Destination

To set the target for diverted calls, send 9 in the first byte, 6 in the second and the following bytes should be a character string representing the divert destination extension. For example, to set the divert destination to extension 236, send the following buffer:

```
unsigned char buf[6];
int len = 6;
buf[0] = 9; // The first two bytes are devspecific constants
buf[1] = 6;
buf[2] = '2';
buf[3] = '3';
buf[4] = '6';
buf[5] = 0; // Don't forget the null terminator
```

Message Waiting Lamp

Some phones have lights that are lit when the user has voicemail messages waiting for them. The number of messages waiting can be controlled by a devspecific command. Zero messages will extinguish the lamp. One or more messages will light the lamp. Send the following buffer to lineDevSpecific:

```
unsigned char buf[21];
int len = 21;
buf[0] = 9; // Shortcode
buf[1] = 73; // Set MWL
sprintf(&(buffer[2]), ";Mailbox Msgs=%d", num);
// Where num is the number of messages
```

Note

- The IP Office server or other IP Office applications may also control the message waiting lamp.

Forward (Divert) Settings

The following constants will help with switching divert features on and off:

```
const unsigned char ForwardAllOn = 0;
const unsigned char ForwardAllOff = 1;
const unsigned char ForwardBusyOn = 2;
const unsigned char ForwardBusyOff = 3;
const unsigned char ForwardNoAnswerOn = 4;
const unsigned char ForwardNoAnswerOff = 5;
const unsigned char DoNotDisturbOn = 7;
const unsigned char DoNotDisturbOff = 8;
```

A buffer that uses any of these constants should be three bytes in length and should begin with a 9. For example, the following code will switch the line to 'Do Not Disturb':

```
unsigned char buf[3];
int len = 3;
buf[0] = 9;
buf[1] = DoNotDisturbOn;
buf[2] = 0;
```

Group Enable and Disable

Send the following buffer to enable the user's membership in the group with extension *groupnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9;
buf[1] = 76;
sprintf((char*)&buf[2], "%d", groupnum);
```

Send the following buffer to disable the user's membership in the group with extension *groupnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9;
buf[1] = 77;
sprintf((char*)&buf[2], "%d", groupnum);
```

In both cases (disabling and enabling group membership), you may elect to disable or enable *all* group membership by omitting the group number and placing a zero in its place (ie. buf[2] = 0).

Note

- You can only enable and disable a user from groups that the user is a member of, as configured in Manager.

Intrude

Send the following buffer to intrude upon another callers call. The call party to be intruded upon is identified by the integer *extnnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9; // Shortcode
buf[1] = 83; // Intrude
sprintf((char*)&buf[2], "%d", extnnum);
```

Listen

Send the following buffer to listen to another callers call. The call party to be listened to is identified by the integer *extnnum*.

```
unsigned char buf[10];
int len = 10;
buf[0] = 9; // Shortcode
buf[1] = 100; // Listen
sprintf((char*)&buf[2], "%d", extnnum);
```

lineDial

This function is used to dial a number on an existing call. It can be used as part of a supervised transfer (see lineSetupTransfer). Country code is ignored.

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

lineDrop

Hangs up a call. UserUserInfo is not supported and will be ignored.

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

lineGenerateDigits

Call this function to generate DTMF digits on the call. The user does not need to be a WAV user and the wave driver does not need to be involved in the call. A LINE_GENERATE message will be sent to the application when the generation is finished. The only dwDigitMode supported is LINEDIGITMODE_DTMF.

```
LONG

WINAPI

lineGenerateDigits(

HCALL hCall,
DWORD dwDigitMode,
LPCSTR lpszDigits,
DWORD dwDuration
);
```

lineGenerateTone

This function can be used to generate a beep on the line. The line must be a WAV user and the wave driver must be involved in the call. The only supported value for dwToneMode is LINETONEMODE_BEEP. As we do not support custom tones, dwNumTones should be zero.

```
LONG

WINAPI

lineGenerateTone(

HCALL hCall,
DWORD dwToneMode,
DWORD dwDuration,
DWORD dwNumTones,
LPLINEGENERATETONE const lpTones
);
```

lineGetAddressCaps

Retrieves the telephony capabilities of a particular address for a particular line. The capabilities are returned in the LINEADDRESSCAPS structure. See LINEADDRESSCAPS in the TAPI structures section for details.

IP Office lines always have a single address.

```
LONG

WINAPI

lineGetAddressCaps(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAddressID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
LPLINEADDRESSCAPS lpAddressCaps
);
```

lineGetAddressID

This function is used to map a phone number (address) assigned to a line device back to its *dwAddressID* in the range zero to the number of addresses minus one returned in the line's device capabilities (LINEDEVCAPS). Given that dwNumAddresses in LINEDEVCAPS is 1, this function will always return 0 in the DWORD pointed to by lpdwAddressID.

```
LONG

WINAPI

lineGetAddressID(

HLINE hLine,
LPDWORD lpdwAddressID,
DWORD dwAddressMode,
LPCSTR lpsAddress,
DWORD dwSize
);
```

lineGetAddressStatus

This function allows an application to query the specified address for its current status. See LINEADDRESSSTATUS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetAddressStatus(

HLINE hLine,
DWORD dwAddressID,
LPLINEADDRESSSTATUS lpAddressStatus
);
```

lineGetAppPriority

Retrieve your applications priority.

```
LONG

WINAPI

lineGetAppPriority(

LPCSTR lpszAppFilename,
DWORD dwMediaMode,
LPLINEEXTENSIONID lpExtensionID,
DWORD dwRequestMode,
LPVARSTRING lpExtensionName,
LPDWORD lpdwPriority
);
```

lineGetCallInfo

Obtain fixed information about the specified call. See LINECALLINFO structure for details.

```
LONG

WINAPI

lineGetCallInfo(

HCALL hCall,
LPLINECALLINFO lpCallInfo
);
```

lineGetCallStatus

This function retrieves a LINECALLSTATUS structure relating to an existing call. See LINECALLSTATUS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetCallStatus(

HCALL hCall,
LPLINECALLSTATUS lpCallStatus
);
```

lineGetDevCaps

Call this function to retrieve the LINEDEVCAPS structure. See LINEDEVCAPS in the TAPI structures section for details.

```
LONG

WINAPI

lineGetDevCaps(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
LPLINEDEVCAPS lpLineDevCaps
);
```

lineGetID

Get the ID for a line when dwSelect is LINECALLSELECT_LINE.

```
LONG

WINAPI

lineGetID(

HLINE hLine,
DWORD dwAddressID,
HCALL hCall,
DWORD dwSelect,
LPVARSTRING lpDeviceID,
LPCSTR lpszDeviceClass
);
```

lineGetLineDevStatus

The lineGetLineDevStatus returns a device specific buffer. The devspecific buffer contains the following information:

```
LONG

WINAPI

lineGetLineDevStatus(

HLINE hLine,
LPLINEDEVSTATUS lpLineDevStatus
);
```

Byte	Contains	Comment
0..n	Phone Extension	This is the line number being monitored, as a character string (eg. "217")
n+1	0	Null terminator for the string above.
n+2	Forward on busy	1 if the phone is set to forward on busy, 0 otherwise.
n+3	Forward on no answer	1 if the phone is set to forward on no answer, 0 otherwise.
n+4	Forward unconditional	1 if the phone is set to forward all.
n+5	Forward hunt group flag	1 if the phone is set to forward hunt group calls.
n+6	Do Not Disturb	1 if the phone is set to DND
n+7	Outgoing call bar flag	1 if the phone is barred from making external calls
n+8	Call waiting on flag	1 if call waiting is enabled for this phone
n+9	Voicemail on flag	1 if voicemail is enabled for this phone
n+10	Voicemail ring-back flag	1 if voicemail ringback is enabled for this phone
n+11	Number of read voicemail messages	The number of read messages.
n+12	Number of unread voicemail messages	The number of voicemail messages waiting for the user.
n+13	Outside call sequence number	Type of ring for external calls.
n+14	Inside call sequence number	Type of ring for internal calls.
n+15	Ring back sequence number	Type of ring for ringback calls.
n+16	No answer timeout period	Number of seconds the phone will ring before following the *no answer* action, e.g. forward on no answer, divert to voicemail.
n+17	Wrap up time period	Number of seconds the phone will remain unable to accept calls following a call.
n+18	Can intrude flag	1 if this phone can intrude upon calls.
n+19	Cannot be intruded upon flag	1 if this phone cannot be intruded upon.

n+20	X directory flag	1 if this user does not appear in the internal directory.
n+21	Force login flag	in the logged-out state on power up and therefore a user must log in.
n+22	Forced account code flag	1 if this phone is forced to provide a valid account code when making external calls.
n+23	Login code flag	1 if this user has a login code configured.
n+24	System phone flag	1 if this is a system phone
n+25	Absent message id	The id of the absent message
n+26	Absent message set flag	1 if the absent message with the id in the previous field is displayed on the phone.
n+27	Voicemail email mode	1 if voicemail email mode is enabled
n+28..m	Extn	The user extension, which may be different to the phone extension.
m+1	0	Null terminator for Extn string above.
m+2..p	locale	The locale of the user.
p+1	0	Null terminator for locale
p+2..q	Forward destination	The number that this phone is set to divert to
q+1	0	Null terminator for destination above.
q+2..r	Follow me number	All calls are redirected to this number.
r+1	0	
r+2..s	Absent text	The absent text defined for this phone.
s+1	0	
s+2..t	Do not disturb exception list	A list of numbers that are permitted to ring the phone while it is in the Do Not Disturb state. Each number is a null-terminated string. The last number in the list is terminated by two nulls (field "t+1" represents the second of these two nulls). If the list is empty then the data will contain just a single null (represented by t+1).
t+1	0	
t+2..u	Forward on busy number	The number that calls will divert to when this phone is busy.
u+1	0	
u+2	User's priority	This priority will be associated with all calls made by this user.
u+3	Group membership	This byte contains the number of groups that the user is currently enabled in.
u+4	Groups out of time	Number of groups that the user is a member of that are currently outside their time profile
u+5	Disabled groups	Number of groups the user is currently disabled from
u+6	Groups out of service	Number of groups that the user is a member of that

		are currently out of service
u+7	Night service groups	Number of groups that the user is a member of that are currently on night service

lineHandoff

The lineHandoff function gives ownership of the specified call to another application.

```
LONG

WINAPI

lineHandoff(

HCALL hCall,
LPCSTR lpszFileName,
DWORD dwMediaMode
);
```

lineHold

This function holds an active call.

```
LONG

WINAPI

lineHold(

HCALL hCall
);
```

lineInitializeEx

This is the first TAPI function that should be called to initialise TAPI. The lpdwAPIVersion parameter should be set to at least 0x00020000.

```
LONG

WINAPI

lineInitializeEx(

LPHLINEAPP lphLineApp,
HINSTANCE hInstance,
LINECALLBACK lpfnCallback,
LPCSTR lpszFriendlyAppName,
LPDWORD lpdwNumDevs,
LPDWORD lpdwAPIVersion,
LPLINEINITIALIZEEXPARAMS lpLineInitializeExParams
);
```

lineMakeCall

This function makes a call. See section on call parameters at the end of the TAPI functions chapter.

```
LONG

WINAPI

lineMakeCall(

HLINE hLine,
LPHCALL lphCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode,
LPLINECALLPARAMS const lpCallParams
);
```

lineMonitorDigits

Call this function to enable the detection of DTMF digits. This function only works when the IP Office wave driver is involved in the call and the user is a WAV user (see the "WAV users" section). Detection is done by analysing media samples in the WAV driver. When a DTMF tone is detected a LINE_MONITORDIGITS message is sent to the application. dwDigitModes can be LINEDIGITMODE_DTMF and/or LINEDIGITMODE_DTMFEND. Call lineMonitorDigits with a dwDigitMode of zero to cancel DTMF digit detection.

```
LONG

WINAPI

lineMonitorDigits(

HCALL hCall,
DWORD dwDigitModes
);
```

lineMonitorTone

This function, like the one above, requires that the wave driver be involved in the call. Furthermore, it can only be used to detect silence. The frequencies in the LINEMONITORTONE structure pointed to by lpToneList must all be zero. If silence is detected, a LINE_MONITORTONE message is sent to the application. Call lineMonitorTone with lpToneList set to NULL to cancel silence detection.

```
LONG

WINAPI

lineMonitorTone(

HCALL hCall,
LPLINEMONITORTONE const lpToneList,
DWORD dwNumEntries
);
```

lineNegotiateAPIVersion

This function should be called immediately after lineInitializeEx to ensure that correct TAPI notifications are sent to your application. It must be called for every line that your application uses.

```
LONG

WINAPI

lineNegotiateAPIVersion(

HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAPILowVersion,
DWORD dwAPIHighVersion,
LPDWORD lpdwAPIVersion,
LPLINEEXTENSIONID lpExtensionID
);
```

lineOpen

This function opens a line device.

dwMediaModes should be set to LINEMEDIAMODE_INTERACTIVEVOICE for ISDN / T1 and LINEMEDIAMODE_UNKNOWN for Analogue trunks. You can specify both to handle calls from both trunk types.

```
LONG

WINAPI

lineOpen(

HLINEAPP hLineApp,
DWORD dwDeviceID,
LPHLINE lphLine,
DWORD dwAPIVersion,
DWORD dwExtVersion,
DWORD dwCallbackInstance,
DWORD dwPrivileges,
DWORD dwMediaModes,
LPLINECALLPARAMS const lpCallParams
);
```

Note

- If an attempt is made to open a line that is associated with a Wave user and no there is no Wave User license installed in IP Office, lineOpen will return LINEERR_RESOURCEUNAVAIL. For an explanation of Wave Users, see WAV Users.

linePark

This function parks a call. Only park mode LINEPARKMODE_DIRECTED is supported. The park address may be any alphanumeric string, however, only numeric digits can be entered from a telephone, so you may want to restrict your park addresses to numeric strings. The four default park addresses that appear in Phone Manager and eConsole are 1, 2, 3 and 4. You should use these numbers if you want parked calls to be unparked using these applications using the default configuration.

```
LONG

WINAPI

linePark(

HCALL hCall,
DWORD dwParkMode,
LPCSTR lpszDirAddress,
LPVARSTRING lpNonDirAddress
);
```

lineRedirect

The lineRedirect function redirects the specified offering call to the specified destination address. Country code is ignored.

```
LONG

WINAPI

lineRedirect(

HCALL hCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode
);
```

lineRemoveFromConference

Removes the call from the conference.

```
LONG

WINAPI

lineRemoveFromConference(

HCALL hCall
);
```

lineSetAppPriority

Call this to indicate your applications priority.

```
LONG

WINAPI

lineSetAppPriority(

LPCSTR lpszAppFilename,
DWORD dwMediaMode,
LPLINEEXTENSIONID lpExtensionID,
DWORD dwRequestMode,
LPCSTR lpszExtensionName,
DWORD dwPriority
);
```

lineSetAppSpecific

This function enables an application to set the application-specific field of the specified call's call-information record.

```
LONG

WINAPI

lineSetAppSpecific(

HCALL hCall,
DWORD dwAppSpecific
);
```

lineSetCallPrivilege

Call this to change your applications ownership rights to a particular call.

```
LONG

WINAPI

lineSetCallPrivilege(

HCALL hCall,
DWORD dwCallPrivilege
);
```

lineSetStatusMessages

This function enables the application to state which notification messages it requires. Typically, dwLineStates is set to LINEDEVSTATE_ALL, and dwAddressStates is set to LINEADDRESSSTATE_ALL.

```
LONG

WINAPI

lineSetStatusMessages(

HLINE hLine,
DWORD dwLineStates,
DWORD dwAddressStates
);
```

lineSetupTransfer

This function is called to create a consultation call in order to perform a supervised transfer. The call that is to be transferred must exist already. The call may be either active or on hold when this function is called. If the call is active it will be put on hold by this function. Call lineDial to ring the party that is to be transferred to. Call lineCompleteTransfer to complete the transfer.

```
LONG

WINAPI

lineSetupTransfer(

HCALL hCall,
LPHCALL lphConsultCall,
LPLINECALLPARAMS const lpCallParams
);
```

lineShutdown

Finish using TAPI line functions. Normally called as your application closes down.

```
LONG
WINAPI
lineShutdown(
HLINEAPP hLineApp
);
```

lineSwapHold

This function puts the current active call on hold and retrieves the held call.

```
LONG
WINAPI
lineSwapHold(
HCALL hActiveCall,
HCALL hHeldCall
);
```

lineUnhold

This function retrieves a held call. If the line is ringing a third party or has an active call with a third party, when this function is called, then the ringing/active call will be dropped before the held call is retrieved.

```
LONG
WINAPI
lineUnhold(
HCALL hCall
);
```

lineUnpark

This function retrieves a parked call. dwAddressID should be 0 because IP Office lines only have one address. lpszDestAddress should be the same identifier that was used to park the call (see linePark).

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

TAPI Structures

LINEADDRESSCAPS

This structure is returned by the lineGetAddressCaps function. The following table indicates the values that are returned for lines that relate to the IP Office TAPI driver.

Note

- Not all members of this structure are listed. For full information on the LINEADDRESSCAPS, see the Microsoft TAPI documentation.

Member	Description / Value
dwLineDeviceID	The ID of the line to which this address relates.
dwDevSpecificSize	No extra information specific to the device is passed.
dwDevSpecificOffset	0
dwAddressSharing	LINEADDRESSSHARING_PRIVATE
dwAddressStates	0
dwCallInfoStates	Returns the possible call info states which are:- LINECALLINFOSTATE_CALLID LINECALLINFOSTATE_RELATEDCALLID LINECALLINFOSTATE_NUMOWNERINCR LINECALLINFOSTATE_NUMOWNEDECR LINECALLINFOSTATE_NUMMONITORS LINECALLINFOSTATE_CALLERID LINECALLINFOSTATE_CALLEDID LINECALLINFOSTATE_REDIRECTIONID LINECALLINFOSTATE_REDIRECTINGID LINECALLINFOSTATE_DISPLAY LINECALLINFOSTATE_MONITORMODES LINECALLINFOSTATE_CALLDATA
dwCallerIDFlags	Returns the possible caller ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwCalledIDFlags	Returns the possible called ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA

	LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwConnectedIDFlags	Returns the possible connected ID flags which are:- LINECALLPARTYID_NAME LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwRedirectionIDFlags	Returns the possible redirection ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwRedirectingIDFlags	Returns the possible redirecting ID flags which are:- LINECALLPARTYID_BLOCKED LINECALLPARTYID_OUTOFAREA LINECALLPARTYID_NAME LINECALLPARTYID_ADDRESS LINECALLPARTYID_UNKNOWN LINECALLPARTYID_UNAVAIL
dwCallStates	Returns the possible call states which are:- LINECALLSTATE_IDLE (the call no longer exists) LINECALLSTATE_OFFERING (a new call has arrived) LINECALLSTATE_ACCEPTED (the call has been claimed by a application) LINECALLSTATE_DIALTONE (the caller hears dial tone) LINECALLSTATE_DIALING (the switch is receiving dialling information) LINECALLSTATE_RINGBACK (the caller hears ringing) LINECALLSTATE_BUSY (the caller hears the busy signal) LINECALLSTATE_CONNECTED (the caller has been connected end to end) LINECALLSTATE_PROCEEDING (dialling has completed but the call has not yet been connected) LINECALLSTATE_ONHOLD (the call is on hold) LINECALLSTATE_CONFERENCED (the call is on

	a conference) LINECALLSTATE_ONHOLDPENDCONF (the call is on hold before being conferenced) LINECALLSTATE_ONHOLDPENDTRANSFER (the call is on hold before being transferred) LINECALLSTATE_DISCONNECTED (The other end has dropped the call) LINECALLSTATE_UNKNOWN (the call state is unknown)
dwDialToneModes	Returns the possible dial tone mode of LINEDIALTONEMODE_UNAVAIL
dwBusyModes	Returns the possible busy modes of LINEBUSYMODE_UNAVAIL
dwSpecialInfo	Returns the possible special info of LINESPECIALINFO_UNAVAIL
dwDisconnectModes	Returns the possible disconnect modes which are:- LINEDISCONNECTMODE_NORMAL LINEDISCONNECTMODE_REJECT LINEDISCONNECTMODE_PICKUP LINEDISCONNECTMODE_FORWARDED LINEDISCONNECTMODE_BUSY LINEDISCONNECTMODE_NOANSWER LINEDISCONNECTMODE_BADADDRESS LINEDISCONNECTMODE_UNREACHABLE LINEDISCONNECTMODE_CONGESTION LINEDISCONNECTMODE_INCOMPATIBLE LINEDISCONNECTMODE_UNAVAIL LINEDISCONNECTMODE_NODIALTONE LINEDISCONNECTMODE_QOSUNAVAIL LINEDISCONNECTMODE_BLOCKED LINEDISCONNECTMODE_DONOTDISTURB
dwMaxNumActiveCalls	The maximum number of active calls: 1
dwMaxNumOnHoldCalls	The maximum number of calls on hold: 9
dwMaxNumOnHoldPendingCalls	The maximum number of calls on hold pending: 9
dwMaxNumConference	The maximum number of conference calls: 9
dwMaxNumTransConf	The maximum number of transferred conference calls: 9
dwAddrCapFlags	Returns the possible address cap flags which are: LINEADDRCAPFLAGS_FWDNUMRINGS LINEADDRCAPFLAGS_DIALED

	LINEADDRCAPFLAGS_TRANSFERHELD LINEADDRCAPFLAGS_TRANSFERMAKE LINEADDRCAPFLAGS_CONFERENCEHELD LINEADDRCAPFLAGS_CONFERENCEMAKE LINEADDRCAPFLAGS_FWDSTATUSVALID
dwCallFeatures	Returns the possible call features which are:- LINECALLFEATURE_ADDTOCONF LINECALLFEATURE_ANSWER LINECALLFEATURE_BLINDTRANSFER LINECALLFEATURE_COMPLETETRANSF LINECALLFEATURE_DIAL LINECALLFEATURE_DROP LINECALLFEATURE_GENERATEDIGITS LINECALLFEATURE_HOLD LINECALLFEATURE_PARK LINECALLFEATURE_REDIRECT LINECALLFEATURE_REMOVEFROMCONF LINECALLFEATURE_SETUPTRANSFER LINECALLFEATURE_SWAPHOLD LINECALLFEATURE_UNHOLD LINECALLFEATURE_SETCALLDATA
dwRemoveFromConfCaps	Returns the possible remove from conference caps which is LINEREMOVEFROMCONF_ANY.
dwRemoveFromConfState	Returns the possible remove from conference state which is LINECALLSTATE_ONHOLD.
dwTransferModes	Returns the possible transfer modes which are: LINETRANSFERMODE_TRANSFER LINETRANSFERMODE_CONFERENCE
dwParkModes	Returns the possible park mode of LINEPARKMODE_DIRECTED.
dwForwardModes	Returns the possible forward modes which are:- LINE FORWARDMODE_UNCOND LINE FORWARDMODE_UNCONDEXTERNAL LINE FORWARDMODE_UNCONDSPECIFIC LINE FORWARDMODE_BUSY LINE FORWARDMODE_BUSYINTERNAL LINE FORWARDMODE_BUSYEXTERNAL LINE FORWARDMODE_BUSYSPECIFIC LINE FORWARDMODE_NOANSW LINE FORWARDMODE_NOANSWINTERNAL

	LINE FORWARDMODE_NOANSWEXTERNAL LINE FORWARDMODE_NOANSWSPECIFIC LINE FORWARDMODE_BUSYNA LINE FORWARDMODE_BUSYNAINTERNAL LINE FORWARDMODE_BUSYNAEXTERNAL LINE FORWARDMODE_BUSYNASPECIFIC
dwMaxForwardEntries	The maximum number of forwarded entries: 10
dwMaxSpecificEntries	The maximum number of specific entries: 10
dwMinFwdNumRings	The minimum forward number of rings: 1
dwMaxFwdNumRings	The maximum forward number of ring: 99
dwMaxCallCompletions	0
dwCallCompletionConds	0
dwCallCompletionModes	0
dwNumCompletionMessages	0
dwCompletionMsgTextEntrySize	0
dwCompletionMsgTextSize	0
dwCompletionMsgTextOffset	0
dwAddressFeatures	Return the possible address features which are:- LINEADDRFEATURE_FORWARD LINEADDRFEATURE_MAKECALL LINEADDRFEATURE_SETUPCONF LINEADDRFEATURE_UNPARK LINEADDRFEATURE_FORWARDFWD LINEADDRFEATURE_FORWARDDND
dwPredictiveAutoTransferStates	0
dwNumCallTreatments	0
dwCallTreatmentListSize	0
dwCallTreatmentListOffset	0
dwDeviceClassesSize	0
dwDeviceClassesOffset	0
dwMaxCallDataSize	The maximum call data size: 127
dwCallFeatures2	0
dwMaxNoAnswerTimeout	0

dwConnectedModes	0
dwOfferingModes	0
dwAvailableMediaModes	0

LINEADDRESSSTATUS

This structure is returned by lineGetAddressStatus.

Note

- Not all members of this structure are listed. For full information on the LINEADDRESSSTATUS, see the Microsoft TAPI documentation.

Member	Description / Value
dwNumInUse;	Always 1
dwNumActiveCalls;	Reflects the number of active calls.
dwNumOnHoldCalls;	Always 0
dwNumOnHoldPendCalls;	Always 0
dwAddressFeatures;	Indicates the capabilities which are:- LINEADDRFEATURE_MAKECALL LINEADDRFEATURE_SETUPCONF LINEADDRFEATURE_UNPARK
dwNumRingsNoAnswer;	5
dwForwardNumEntries;	Always 0
dwForwardSize; /	Always 0
dwForwardOffset;	Always 0
dwTerminalModesSize;	Always 0
dwTerminalModesOffset;	Always 0
dwDevSpecificSize;	Always 0

LINECALLINFO

This structure is returned by lineGetCallInfo.

Note

- Not all members of this structure are listed. For full information on the LINECALLINFO, see the Microsoft TAPI documentation.

Member	Description / Value
dwAddressID	Always 0
dwBearerMode	Returns the possible bearer mode which is:- LINEBEARERMODE_VOICE
dwRate	64000
dwMediaMode	Returns the possible media mode which is :- LINEMEDIAMODE_INTERACTIVEVOICE
dwAppSpecific	Set by application.
dwCallID	Call ID
dwCallParamFlags	Returns the possible call parameter flags which is:-LINECALLPARAMFLAGS_IDLE
dwCallStates	Returns the possible call states which are:- LINECALLSTATE_IDLE LINECALLSTATE_OFFERING LINECALLSTATE_DIALTONE LINECALLSTATE_DIALING LINECALLSTATE_RINGBACK LINECALLSTATE_BUSY LINECALLSTATE_CONNECTED LINECALLSTATE_PROCEEDING LINECALLSTATE_ONHOLD LINECALLSTATE_CONFERENCED LINECALLSTATE_ONHOLDPENDCONF LINECALLSTATE_ONHOLDPENDTRANSFER LINECALLSTATE_DISCONNECTED LINECALLSTATE_UNKNOWN
dwMonitorMediaModes	0
dwCountryCode	0
dwTrunk	0xFFFFFFFF
dwCommentSize	0
dwCommentOffset	0
dwUserUserInfoSize	0

dwUserUserInfoOffset	0
dwHighLevelCompSize	0
dwHighLevelCompOffset	0
dwLowLevelCompSize	0
dwLowLevelCompOffset	0
dwChargingInfoSize	0
dwChargingInfoOffset	0
dwTerminalModesSize	0
dwTerminalModesOffset	0
dwCallDataOffset	0
dwSendingFlowspecSize	0
dwSendingFlowspecOffset	0
dwReceivingFlowspecSize	0
dwReceivingFlowspecOffset	0
dwCallerIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwCalledIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwConnectedIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwRedirectionIDAddressType	0 – only valid for TAPI Version 3.0 and above
dwRedirectingIDAddressType	0 – only valid for TAPI Version 3.0 and above

LINECALLPARAMS

The following parameters are recognized in the LINECALLPARAMS structure that can be passed to lineMakeCall and lineSetupTransfer:

Note

- Not all members of this structure are listed. For full information on the LINECALLPARAMS, see the Microsoft TAPI documentation.

Member	Description/Value
dwCallParamFlags	Set this to zero for normal use or enter LINEBEARERMODE_VOICE if you wish to conceal the caller line identifier on the call.
dwCalledPartyOffset	Can be used to set the called party identifier.
dwCallingPartyIDOffset	Can be used to set the calling party identifier.

LINECALLSTATUS

This structure is returned by the lineGetCallStatus function.

Note

- Not all members of this structure are listed. For full information on the LINECALLSTATUS, see the Microsoft TAPI documentation.

Member	Description/Value
dwCallState	Returns one of the following states: LINECALLSTATE_IDLE (The call no longer exists) LINECALLSTATE_OFFERING (a new call has arrived) LINECALLSTATE_ACCEPTED (the call has been claimed by an application) LINECALLSTATE_DIALTONE (the caller hears a dial tone) LINECALLSTATE_DIALING (the switch is receiving dialling information) LINECALLSTATE_RINGBACK (the caller hears ringing) LINECALLSTATE_BUSY (the caller hears the busy signal) LINECALLSTATE_CONNECTED (the call has been connected end to end) LINECALLSTATE_PROCEEDING (dialling has completed but the call has not yet been connected) LINECALLSTATE_ONHOLD (the call is on hold) LINECALLSTATE_CONFERENCED (the call is on a conference) LINECALLSTATE_ONHOLDPENDCONF (the call is on hold before being conferenced) LINECALLSTATE_ONHOLDPENDTRANSFER (the call is on hold before being transferred) LINECALLSTATE_DISCONNECTED (the other end has dropped the call) LINECALLSTATE_UNKNOWN (the call state is unknown)
dwCallStateMode	Always zero.
dwCallPrivilege	The applications privilege for this call.
dwCallFeatures	The call features available for the call state indicated by dwCallState. TAPI specifices all possible features, however, only those that appear in dwCallFeatures in the LINEADDRESSCAPS structure can be used.
dwDevSpecificSize	0
dwDevSpecificOffset	0
dwCallFeatures2	0
tStateEntryTime	Zeros

LINEDEVCAPS

This structure is returned by the lineGetDevCaps function. The comments below indicate the values that will be returned for lines that relate to the IP Office TAPI driver.

Note

- Not all members of this structure are listed. For full information on the LINEDEVCAPS, see the Microsoft TAPI documentation.

Member	Description / Value
dwProviderInfoSize	Indicates the Provider Name, i.e. the name of the TSP.
dwSwitchInfoSize	0
dwPermanentLineID	Unique identifier assigned by Windows.
dwLineNameSize	Indicates the Line Name.
dwStringFormat	Returns the string format which is:- STRINGFORMAT_ASCII
dwAddressModes	Returns the address mode which is:- LINEADDRESSMODE_ADDRESSID
dwNumAddresses	1
dwBearerModes	Returns the bearer modes which are: LINEBEARERMODE_VOICE LINEBEARERMODE_SPEECH
dwMaxRate	0
dwMediaModes	Returns the media mode which is: LINEMEDIAMODE_INTERACTIVEVOICE
dwGenerateToneModes	Returns the generate tone mode which is:- LINETONEMODE_BEEP
dwGenerateToneMaxNumFreq	0
dwMonitorToneMaxNumFreq	1
dwMonitorToneMaxNumEntries	1
dwGatherDigitsMinTimeout	0
dwGatherDigitsMaxTimeout	0
dwMedCtlDigitMaxListSize	0
dwMedCtlMediaMaxListSize	0
dwMedCtlToneMaxListSize	0
dwMedCtlCallStateMaxListSize	0
dwDevCapFlags	Returns the dev cap flags which are:- LINEDEVCAPFLAGS_CLOSEDROP

	LINEDEVCAPFLAGS_DIALBILLING LINEDEVCAPFLAGS_DIALQUIET LINEDEVCAPFLAGS_DIALDUALTONE
dwMaxNumActiveCalls	9
dwAnswerMode	Returns the answer mode which is:- LINEANSWERMODE_NONE
dwRingModes	1
dwLineStates	Returns the line state which is:- LINEDEVSTATE_RINGING LINEDEVSTATE_CONNECTED LINEDEVSTATE_DISCONNECTED LINEDEVSTATE_INSERVICE LINEDEVSTATE_OUTOFSERVICE LINEDEVSTATE_OPEN LINEDEVSTATE_CLOSE LINEDEVSTATE_REINIT LINEDEVSTATE_TRANSLATECHNGE LINEDEVSTATE_REMOVED
dwUUIAcceptSize	0
dwUUIAnswerSize	100
dwUUIMakeCallSize	100
dwUUIDropSize	100
dwUUISendUserUserInfoSize	100
dwUUICallInfoSize	User to User call information size: 100
dwNumTerminals	0
dwTerminalCapsSize	0
dwTerminalCapsOffset	0
dwTerminalTextEntrySize	0
dwTerminalTextSize	0
dwTerminalTextOffset	0
dwDevSpecificSize	0
dwDevSpecificOffset	0
dwLineFeatures	Returns the line feature which is: LINEFEATURE_MAKECALL
dwSettableDevStatus	0
dwDeviceClassesSize	tapi\line

PermanentLineGuide	Only relevant if using TAPI Version 2.2 or higher.
dwAddressTypes	Only relevant if using TAPI Version 3.0 or higher.
ProtocolGuide	Only relevant if using TAPI Version 3.0 or higher.
dwAvailableTracking	Only relevant if using TAPI Version 3.0 or higher.

TAPI Events (Messages)

LINE_APPNEWCALL

A new call has been created.

LINE_CALLINFO

Information has changed in the LINECALLINFO structure.

LINE_CALLSTATE

The state of the call has changed. See dwCallStates in the LINEADDRESSCAPS structure for the list of states supported.

LINE_LINEDEVSTATE

The line device state has changed. The second parameter could be any one of the following:

- LINEDEVSTATE_DEVSPECIFIC - Devspecific information has changed.
- LINEDEVSTATE_CONNECTED, LINEDEVSTATE_DISCONNECTED - The connected state of the line has changed.
- LINEDEVSTATE_OUTOFSERVICE - The TSP has lost communication with the switch. This line is now out of service.
- LINEDEVSTATE_INSERVICE - The TSP had lost connection to the switch but has now recovered and the line is back in service.
- LINEDEVSTATE_RINGING - The switch has detected that the caller's phone is ringing.

LINE_DEVSPECIFIC

Notifies the application about device-specific events occurring on a line, address, or call. This message prompts the application to call lineGetLineDevStatus and analyse the devspecific buffer for changes.

LINE_ADDRESSSTATE

The status of an address has changed on a line that is currently open by the application.

TAPI 3.0 Reference

TAPI

TAPI

The TAPI object is created by CoCreateInstance. All other TAPI 3.0 objects are created by TAPI 3.0 itself.

ITTAPI

The ITTAPI interface is the base interface for the TAPI object.

Initialize

This is the first TAPI function that should be called to initialise TAPI.

```
HRESULT
Initialize();
```

Shutdown

Shuts down a TAPI session. Normally called as your applications closes down.

```
HRESULT
Shutdown();
```

EnumerateAddresses

This method enumerates the addresses that are currently available.

```
HRESULT
EnumerateAddresses ( IEnumAddress **ppEnumAddress );
```

RegisterCallNotifications

Sets which new call notifications an application will receive. The application must call the method for each address, indicating media type or types it can handle and specifying the privileges it requests.

```
HRESULT RegisterCallNotifications(
ITAddress *pAddress,
VARIANT_BOOL fMonitor,
VARIANT_BOOL fOwner,
long lMediaTypes,
long lCallbackInstance,
long *plRegister
);
```

put_EventFilter

The **put_EventFilter** method sets the event filter mask

```
HRESULT
put_EventFilter ( long lFilterMask );
```

Address

Address

The Address object represents an entity that can make or receive calls.

ITAddress

The interface is the base interface for the Address object.

get_AddressName

Gets the displayable name of the address.

```
HRESULT
get_AddressName (BSTR *ppName );
```

get_DialableAddress

The **get_DialableAddress** method gets the **BSTR,** which can be used to connect to this address.

```
HRESULT
get_DialableAddress (
BSTR *pDialableAddress
);
```

get_ServiceProviderName

The **get_ServiceProviderName** method gets the name of the Telephony Service Provider (TSP) that supports this address: for example, Unimdm.tsp for the Unimodem service provider or H323.tsp for the H323 service provider.

```
HRESULT
get_ServiceProviderName (
BSTR *ppName
);
```

CreateCall

The **CreateCall** method creates a new Call object that can be used to make an outgoing call and returns a pointer to the object's **ITBasicCallControl** interface.

```
HRESULT
CreateCall (
BSTR *pDialableAddress,
Long 1AddressType,
Long 1MediaTypes,
ITBasicCallControl **ppCall
);
```

IEnumAddress

Provides COM-standard enumeration methods for the ITAddress interface.

Next

The **Next** method gets the next specified number of elements in the enumeration sequence.

```
HRESULT
Next(
ULONG celt,
ITAddress **ppElements,
ULONG *pceltFetched
);
```

ITMediaSupport

The ITMediaSupport interface provides methods that allow an application to discover the media support capabilities for an Address Object that exposes this interface.

get_MediaTypes

The get_MediaTypes method gets the media type or types supported on the current address.

```
HRESULT
get_MediaTypes (
long *plMediaTypes
);
```

Terminal

Terminal object represents the source or sink of a media stream associated with a call or communications session.

Call

Call

The Call object represents an address's connection between the local address and one or more other addresses.

ITCallInfo

The ITCallInfo interface gets and sets a variety of information concerning a Call object.

get_Address

The **get_Address** method gets a pointer to the **ITAddress** interface of the Address object.

```
HRESULT
get_Address (
ITAddress **ppAddress
);
```

get_CallState

The **get_CallState** method gets a pointer to the current call state, such as CS_IDLE.

```
HRESULT
get_CallState (
CALL_STATE *pCallState
);
```

get_CallInfoString

The **get_CallInfoString** method gets a call information items described by a string, such as the displayable address.

```
HRESULT
get_CallInfoString (
CALLINFO_STRING CallInfoString,
BSTR *ppCallInfoString
);
```

SetCallInfoBuffer

Either by accident or design, TAPI 3.0 (Windows 2000) only allows this function on a call that is in the IDLE state. This has been changed in TAPI 3.1 (Windows XP) which allows call data to be set on calls in the connected state by passing CIB_CALLDATABUFFER as the CallInfoBuffer parameter.

```
HRESULT
SetCallInfoBuffer (
CALLINFO_BUFFER CallInfoBuffer,
DWORD dwSize
BYTE* pCallInfoBuffer
);
```

ITBasicCallControl

The ITBasicCallControl interface is used by the application to connect, answer, and perform basic telephony operations on a call object.

Connect

The Connect method attempts to complete the connection of an outgoing call.

```
HRESULT
Connect(
VARIANT_BOOL fSync
);
```

Answer

The Answer method answers an incoming call. This method can succeed only if the call state is CS_OFFERING.

```
HRESULT
Answer();
```

Disconnect

The Disconnect method disconnects the call. The call state will transition to CS_DISCONNECTED after the method completes successfully.

```
HRESULT
Disconnect(
DISCONNECT_CODE code
);
```

Hold

The **Hold** method places or removes the call from the hold.

```
HRESULT
Hold(
VARIANT_BOOL fHold
);
```

SwapHold

The **SwapHold** method swaps the call (which is active) with the specified call on hold.

```
HRESULT
SwapHold(
ITBasicCallControl *pCall
);
```

ParkDirect

The **ParkDirect** method parks the call at a specified address.

```
HRESULT
ParkDirect(
BSTR pParkAddress
);
```

Unpark

The **Unpark** method gets the call from park.

```
HRESULT
Unpark();
```

BlindTransfer

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

```
HRESULT BlindTransfer(
BSTR pDestAddress
);
```

Transfer

The **Transfer** method transfers the current call to the destination address.

```
HRESULT Transfer(
ITBasicCallControl *pCall,
VARIANT_BOOL fSync
);
```

Finish

The **Finish** method is called on a consultation call to finish a conference or a transfer.

```
HRESULT Finish(
FINISH_MODE finishMode
);
```

Conference

The **Conference** method adds a consultation call to the conference in which the current call is a participant.

```
HRESULT Conference(
ITBasicCallControl *pCall,
VARIANT_BOOL fSync
);
```

RemoveFromConference

The **RemoveFromConference** method removes the call from a conference if it is involved in one.

```
HRESULT RemoveFromConference();
```

ITCallStateEvent

The ITCallStateEvent interface contains methods that retrieve the description of call state events.

get_Cause

The **get_Cause** method gets the cause associated with this event.

```
HRESULT
get_Cause (
CALL_STATE_EVENT_CAUSE *pCEC
);
```

get_State

The **get_State** method gets information on the new call state.

```
HRESULT
get_State (
CALL_STATE *pCallState
);
```

get_Call

The **get_Call** method gets a pointer to the call information interface for the call on which the event has occurred.

```
HRESULT
get_Call
ITCallInfo **ppCallInfo
);
```

ITCallNotificationEvent

The ITCallNotificationEvent interface contains methods that retrieve the description of call notification events.

get_Call

The **get_Call** method returns the **ITCallInfo** interface on which a call event has occurred.

```
HRESULT
get_Call
ITCallInfo **ppCall
);
```

ITCallInfoChangeEvent

The ITCallInfoChangeEvent interface contains methods that retrieve the description of call information change events.

get_Call

The **get_Call** method returns the **ITCallInfo** interface on which call information has changed.

```
HRESULT
get_Call
ITCallInfo **ppCall
);
```

Call Hub

The Call Hub object exposes methods that retrieve information concerning participants in a multi-party call. Call Hubs are not supported by IP Office. Call Hub Events may be received but should be ignored.

TAPI 3 Enumerated Types

CALL_STATE

The CALL_STATE enum is used by the ITCallInfo::get_CallState and ITCallStateEvent::get_State methods.

Member	Value	Description
CS_IDLE	0	The call has been created, but **Connect** has not been called yet. A call can never transition into the idle state.
CS_INPROGRESS	1	**Connect** has been called, and the service provider is working on making a connection. This state is valid only on outgoing calls. This message is optional, because a service provider may have a call transition directly to the connected state.
CS_CONNECTED	2	Call has been connected to the remote end and communication can take place.
CS_DISCONNECTED	3	Call has been disconnected. There are several causes for disconnection. See the table of valid call state transitions below.
CS_OFFERING	4	A new call has appeared, and is being offered to an application. If the application has owner privileges on the call, it can either call **Answer** or **Disconnect** while the call is in the offering state.
CS_HOLD	5	The call is in the hold state.
CS_QUEUED	6	The call is queued.

CALLINFO_STRING

The **CALLINFO_STRING** enum is used by **ITCallInfo** methods that set and get call information involving the use of strings.

Member	Value	Description
CIS_CALLERIDNAME	0	The name of the caller.
CIS_CALLERIDNUMBER	1	The number of the caller.
CIS_CALLEDIDNAME	2	The name of the called location.
CIS_CALLEDIDNUMBER	3	The number of the called location.
CIS_CONNECTEDIDNAME	4	The name of the connected location.
CIS_CONNECTEDIDNUMBER	5	The number of the connected location.
CIS_REDIRECTIONIDNAME	6	The name of the location to which a call has been redirected.
CIS_REDIRECTIONIDNUMBER	7	The number of the location to which a call has been redirected.
CIS_REDIRECTINGIDNAME	8	The name of the location that redirected the call.
CIS_REDIRECTINGIDNUMBER	9	The number of the location that redirected the call.
CIS_CALLEDPARTYFRIENDLYNAME	10	The called party friendly name.
CIS_COMMENT	11	A comment about the call provided by the application that originated the call.
CIS_DISPLAYABLEADDRESS	12	A displayable version of the called or calling address.
CIS_CALLINGPARTYID	13	The identifier of the calling party.

DISCONNECT_CODE

The DISCONNECT_CODE enum is used by the ITBasicCallControl::Disconnect method.

Member	Value	Description
DC_NORMAL	0	The call is being disconnected as part of the normal cycle of the call.
DC_NOANSWER	1	The call is being disconnected because it has not been answered. (For example, an application may set a certain amount of time for the user to answer the call. If the user does not answer, the application can call **Disconnect** with the NOANSWER code.)
DC_REJECTED	2	The user rejected the offered call.

CALL_STATE_EVENT_CAUSE

The CALL_STATE_EVENT_CAUSE enum is returned by the ITCallStateEvent::get_Cause method.

Member	Value	Description
CEC_NONE	0	No call event has occurred.
CEC_DISCONNECT_NORMAL	1	The call was disconnected as part of the normal life cycle of the call (that is, the call was over, so it was disconnected).
CEC_DISCONNECT_BUSY	2	An outgoing call failed to connect because the remote end was busy.
CEC_DISCONNECT_BADADDRESS	3	An outgoing call failed because the destination address was bad.
CEC_DISCONNECT_NOANSWER	4	An outgoing call failed because the remote end was not answered.
CEC_DISCONNECT_CANCELLED	5	An outgoing call failed because the caller disconnected.
CEC_DISCONNECT_REJECTED	6	The outgoing call was rejected by the remote end.
CEC_DISCONNECT_FAILED	7	The call failed to connect for some other reason.

The IP Office Media Service Provider

About the MSP

The IP Office Media Service Provider serves a dual purpose. It providers media streaming capability which allows a TAPI 3 application to send and receive voice data on calls that are present on specific types of users' lines. It also allows an application access to device specific functionality of the IP Office.

Using The MSP

The media service provider interfaces are documented in the MSDN libraries. The DevSpice sample on the SDK CD gives an example of how to use the MSP for media streaming and device specific functionality. The MSP is available to every TAPI address that can be viewed in your TAPI 3 application. Media streaming capabilities are only available to addresses that are specifically named as WAVE users. WAVE users are users with a name that begins with "TAPI:" (Such as "TAPI:201"). You may create as many WAV users as you wish, but each WAVE user will require a wave driver licence instance to enable media streaming to that user.

Using the Device Specific Interfaces

The device specific interfaces are implemented on the Address and Call objects of the MSP. TAPI 3.0 will delegate queries for interfaces it does not recognise to the MSP. If, therefore, you have a pointer to an ITAddress interface, you can call QueryInterface to retrieve a pointer to the ITDivert interface (for example). The following code from the DevSpice sample illustrates:

```
ITDivert* pDivert = NULL;

if( SUCCEEDED( gpAddress->QueryInterface( IID_ITDIVERT,

(void**)&pDivert)))

{

DWORD dwDivertSettings = 0;
if( FAILED( pDivert-> GetDivertSettings(
&dwDivertSettings)))

{
```

The interfaces available from the address object are:

- ITACDAgent
- ITDivert
- ITGroup

The interface available from the Call object is:

- ITPlay

Furthermore, the address object acts as a connection point container for IP Office Private events. The connection point interface is available in the interfaces.h file of the DevSpice sample and is called IPOfficePrivateEvent. Details of these interfaces are given below.

ITACDAgent

IsLoggedIn(void)	Returns S_TRUE if the user is logged in and S_FALSE if the user is logged out.
LogOut(void)	Logs the user off of this line. The user must have "force logon" set in Manager.
Login(BSTR extn)	Logs the user onto the given extension.
CallListen(BSTR extn)	Listens to the call present at the given extension. The user must have the Can Intrude privilege set in Manager.
Intrude(BSTR extn)	Conferences the current user in to the call present at the given extension. The user must have the Can Intrude privilege set in Manager.
SetAccountCode(BSTR extn)	Sets the account code for the current call.

ITGroup

This interface contains functions to take the user in and out of group, as well as to intercept calls that present themselves at other phones in the group.

PickupAny(void)	Equivalent to executing the CallPickupAny shortcode on the user's terminal. See Manager for details.
PickupGroup(void)	Equivalent to executing the CallPickupGroup shortcode on the user's terminal.
PickupExtn(BSTR extn)	Equivalent to executing the CallPickupExtn shortcode on the user's extension.
PickupMembers(BSTR extn)	Equivalent to executing the CallPickupMembers shortcode on the user's extension.
Enable(BSTR groupextn)	Enables the user's membership of the given group. If groupextn is an empty string, the user will be enabled in all groups that he/she is a member of.
Disable(BSTR groupextn)	Disables the user's membership of the given group. If groupextn is an empty string, the user will be disabled in all groups that he/she is a member of.

ITDivert

This interface contains functions for getting and setting the divert flags for the address.

GetDivertAllDestination(BSTR* pDestination)	Gets the current Divert All destination and returns the result in the pDestination value.
SetDivertAllDestination(BSTR dest)	
GetDivertSettings(DWORD* pdwDivertSets)	This function sets bits in the DWORD pointed to by pdwDivertSets to indicate which of the divert settings are currently active. The bits are defined by the IP_OFFICE_DIVERT_SETTINGS enum (described below).
SetForwardAll(VARIANT_BOOL bOn)	Toggles the ForwardAll setting for this user.
SetForwardBusy(VARIANT_BOOL bOn)	Toggles the ForwardBusy setting for this user.
SetForwardNoAnswer(VARIANT_BOOL bOn)	Toggles the ForwardNoAnswer setting for this user.
SetDoNotDisturb(VARIANT_BOOL bOn)	Toggles the DND setting for this user.

The IP_OFFICE_DIVERT_SETTINGS enum is defined as follows:

```
typedef enum
{
IPOFF_FWDALL = 0x01,
IPOFF_FWDBUSY = 0x02,
IPOFF_NOANSWER = 0x04,
IPOFF_DND = 0x08,
IPOFF_DESTINATION = 0x10
} IP_OFFICE_DIVERT_SETTINGS;
```

Therefore, getting a result of 14 (0xe) from GetDivertSettings implies that the user has ForwardBusy, ForwardNoAnswer and DoNotDisturb set. The IPOFF_DESTINATION value is not used by GetDivertSettings, only by the Fire_DivertSettingsChanged function on the IPOfficePrivateEvents interface.

ITPlay

The ITPlay interface is implemented on the MSP Call object. It allows for recording and playing of wave files.

StartPlay(BSTR FileName)	FileName should be the complete path to a wave file to play.
StopPlay()	
StartRecord(BSTR FileName)	FileName should be the complete path to a wave file to record to.
StopRecord()	

Playing and recording can be stopped and started at any time on the call. Recording will use only a single file per call though, and will append to the file if recording is stopped and restarted. It is not advisable to attempt to record and play at the same time. If this is a requirement, recording and playing can be done by selecting terminals onto the call that supply audio data from a file, or record audio data to a file. TAPI 3.1 introduces file-streaming terminals to make this easier.

IPOfficePrivateEvents

This is a connection point interface that the MSP uses to report events on. See the DevSpice sample on how to register for, and handle, private events.

OnUserLogin(void)	Fired when an agent (a user with 'force logon' set in Manager) logs on.
OnUserLogout(void)	Fired when an agent logs out.
OnDivertSettingsChanged(DWORD dwDivertSettings)	Fired when the user changes one of their divert setting flags (such as Do Not Disturb or Divert On Busy) or the Divert All destination. The bits in the dwDivertSettings variable are set using the IP_OFFICE_DIVERT_SETTINGS enum described above.
OnGroupChanged(DWORD dwGroupCount)	Fired when the user enables or disables group membership. The dwGroupCount value gives the number of groups that this user is an enabled member of.
OnVoiceMail(DWORD dwNumMessages)	This is fired when the number of voicemail messages that the user has waiting for them changes. The new value is given in the dwNumMessages parameter.

Using the Media Streaming Capabilities of the MSP

The IP Office MSP handles media streaming to any wave user. In order to do this, you must select terminals onto the streams that the MSP exposes for a call. Details on how to do this are given in the MSDN and an example is shown in the DevSpice sample. It should be noted that IP Office streams are bi-directional, and there is only a single stream per call. This means that both capture and render terminals are accepted on the same stream (but only one of each).

The MSP encapsulates the functionality of the IP Office wave driver. The IP Office wave driver must be installed on each machine that wishes to do media streaming to wave users. If you do not wish to do media streaming, and only wish to monitor wave users using TAPI, you must ensure that the wave driver is not installed, or you will consume wave licence instances for every wave user line you open.

Index

Avaya
Unit 1, Sterling Court
15 - 21 Mundells
Welwyn Garden City
Hertfordshire
AL7 1LZ
England

Tel: +44 (0) 1707 392200
Fax: +44 (0) 1707 376933

Web: http://www.avaya.com/ipoffice/knowledgebase