# IP Office TAPILink Developer's Guide

Release 11.1
Issue 4
May 2022

**Notice**

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

**Documentation disclaimer**

“Documentation” means information published in varying mediums which may include product information, operating instructions and performance specifications that are generally made available to users of products. Documentation does not include marketing materials. Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of Documentation unless such modifications, additions, or deletions were performed by or on the express behalf of Avaya. End User agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

**Link disclaimer**

Avaya is not responsible for the contents or reliability of any linked websites referenced within this site or Documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

**Warranty**

Avaya provides a limited warranty on Avaya hardware and software. Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya’s standard warranty language, as well as information regarding support for this product while under warranty is available to Avaya customers and other parties through the Avaya Support website: https://support.avaya.com/helpcenter/getGenericDetails?detailId=C20091120112456651010 under the link “Warranty & Product Lifecycle” or such successor site as designated by Avaya. Please note that if You acquired the product(s) from an authorized Avaya Channel Partner outside of the United States and Canada, the warranty is provided to You by said Avaya Channel Partner and not by Avaya.

“**Hosted Service**” means an Avaya hosted service subscription that You acquire from either Avaya or an authorized Avaya Channel Partner (as applicable) and which is described further in Hosted SAS or other service description documentation regarding the applicable hosted service. If You purchase a Hosted Service subscription, the foregoing limited warranty may not apply but You may be entitled to support services in connection with the Hosted Service as described further in your service description documents for the applicable Hosted Service. Contact Avaya or Avaya Channel Partner (as applicable) for more information.

**Hosted Service**

THE FOLLOWING APPLIES ONLY IF YOU PURCHASE AN AVAYA HOSTED SERVICE SUBSCRIPTION FROM AVAYA OR AN AVAYA CHANNEL PARTNER (AS APPLICABLE), THE TERMS OF USE FOR HOSTED SERVICES ARE AVAILABLE ON THE AVAYA WEBSITE, HTTPS://SUPPORT.AVAYA.COM/LICENSEINFO UNDER THE LINK “Avaya Terms of Use for Hosted Services” OR SUCH SUCCESSOR SITE AS DESIGNATED BY AVAYA, AND ARE APPLICABLE TO ANYONE WHO ACCESSES OR USES THE HOSTED SERVICE. BY ACCESSING OR USING THE HOSTED SERVICE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE DOING SO (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THE TERMS OF USE. IF YOU ARE ACCEPTING THE TERMS OF USE ON BEHALF A COMPANY OR OTHER LEGAL ENTITY, YOU REPRESENT THAT YOU HAVE THE AUTHORITY TO BIND SUCH ENTITY TO THESE TERMS OF USE. IF YOU DO NOT HAVE SUCH AUTHORITY, OR IF YOU DO NOT WISH TO ACCEPT THESE TERMS OF USE, YOU MUST NOT ACCESS OR USE THE HOSTED SERVICE OR AUTHORIZE ANYONE TO ACCESS OR USE THE HOSTED SERVICE.

**Licenses**

THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, HTTPS://SUPPORT.AVAYA.COM/LICENSEINFO, UNDER THE LINK “AVAYA SOFTWARE LICENSE TERMS (Avaya Products)” OR SUCH SUCCESSOR SITE AS DESIGNATED BY AVAYA, ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AVAYA CHANNEL PARTNER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AVAYA CHANNEL PARTNER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA CHANNEL PARTNER; AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE (“AVAYA”).

Avaya grants You a license within the scope of the license types described below, with the exception of Heritage Nortel Software, for which the scope of the license is detailed below. Where the order documentation does not expressly identify a license type, the applicable license will be a Designated System License as set forth below in the Designated System(s) License (DS) section as applicable. 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 You. “Software” means computer programs in object code, provided by Avaya or an Avaya Channel Partner, whether as stand-alone products, pre-installed on hardware products, and any upgrades, updates, patches, bug fixes, or modified versions thereto. “Designated Processor” means a single stand-alone computing device. “Server” means a set of Designated Processors that hosts (physically or virtually) a software application to be accessed by multiple users. “Instance” means a single copy of the Software executing at a particular time: (i) on one physical machine; or (ii) on one deployed software virtual machine (“VM”) or similar deployment.

**License type(s)**

Designated System(s) License (DS). End User may install and use each copy or an Instance of the Software only: 1) on a number of Designated Processors up to the number indicated in the order; or 2) up to the number of Instances of the Software as indicated in the order, Documentation, or as authorized by Avaya in writing. Avaya may require the Designated Processor(s) to be identified in the order by type, serial number, feature key, Instance, location or other specific designation, or to be provided by End User to Avaya through electronic means established by Avaya specifically for this purpose.

Concurrent User License (CU). End User may install and use the Software on multiple Designated Processors or one or more Servers, so long as only the licensed number of Units are accessing and using the Software at any given time as indicated in the order, Documentation, or as authorized by Avaya in writing. A “Unit” means the unit on which Avaya, at its sole discretion, bases the pricing of its licenses and can be, without limitation, an agent, port or user, an e-mail or voice mail account in the name of a person or corporate function (e.g., webmaster or helpdesk), or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software. Units may be linked to a specific, identified Server or an Instance of the Software.

Cluster License (CL). End User may install and use each copy or an Instance of the Software only up to the number of Clusters as

indicated on the order, Documentation, or as authorized by Avaya in writing with a default of one (1) Cluster if not stated.

Enterprise License (EN). End User may install and use each copy or an Instance of the Software only for enterprise-wide use of an unlimited number of Instances of the Software as indicated on the order, Documentation, or as authorized by Avaya in writing.

Named User License (NU). End User may: (i) install and use each copy or Instance of the Software on a single Designated Processor or Server per authorized Named User (defined below); or (ii) install and use each copy or Instance of the Software on a Server so long as only authorized Named Users access and use the Software as indicated in the order, Documentation, or as authorized by Avaya in writing. “Named User”, means a user or device that has been expressly authorized by Avaya to access and use the Software. At Avaya’s sole discretion, a “Named User” may be, without limitation, designated by name, corporate function (e.g., webmaster or helpdesk), an e-mail or voice mail account in the name of a person or corporate function, or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software.

Shrinkwrap License (SR). End User may install and use the Software in accordance with the terms and conditions of the applicable license agreements, such as “shrinkwrap” or “clickthrough” license accompanying or applicable to the Software (“Shrinkwrap License”) as indicated in the order, Documentation, or as authorized by Avaya in writing.

Transaction License (TR). End User may use the Software up to the number of Transactions as specified during a specified time period and as indicated in the order, Documentation, or as authorized by Avaya in writing. A “Transaction” means the unit by which Avaya, at its sole discretion, bases the pricing of its licensing and can be, without limitation, measured by the usage, access, interaction (between client/server or customer/organization), or operation of the Software within a specified time period (e.g. per hour, per day, per month). Some examples of Transactions include but are not limited to each greeting played/message waiting enabled, each personalized promotion (in any channel), each callback operation, each live agent or web chat session, each call routed or redirected (in any channel). End User may not exceed the number of Transactions without Avaya’s prior consent and payment of an additional fee.

**Heritage Nortel Software**

“Heritage Nortel Software” means the software that was acquired by Avaya as part of its purchase of the Nortel Enterprise Solutions Business in December 2009. The Heritage Nortel Software is the software contained within the list of Heritage Nortel Products located at https://support.avaya.com/LicenseInfo under the link “Heritage Nortel Products” or such successor site as designated by Avaya. For Heritage Nortel Software, Avaya grants Customer a license to use Heritage Nortel Software provided hereunder solely to the extent of the authorized activation or authorized usage level, solely for the purpose specified in the Documentation, and solely as embedded in, for execution on, or for communication with Avaya equipment. Charges for Heritage Nortel Software may be based on extent of activation or use authorized as specified in an order or invoice.

**Copyright**

Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, Hosted Service, or hardware provided by Avaya. All content on this site, the documentation, Hosted Service, and the product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

**Virtualization**

The following applies if the product is deployed on a virtual machine. Each product has its own ordering code and license types. Unless otherwise stated, each Instance of a product must be separately licensed and ordered. For example, if the end user customer or Avaya Channel Partner would like to install two Instances of the same type of products, then two products of that type must be ordered.

**Third Party Components**

“Third Party Components” mean certain software programs or portions thereof included in the Software or Hosted Service may contain software (including open source software) distributed under third party agreements (“Third Party Components”), which contain terms regarding the rights to use certain portions of the Software (“Third Party Terms”). As required, information regarding distributed Linux OS source code (for those products that have distributed Linux OS source code) and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the products, Documentation or on Avaya’s website at: https://support.avaya.com/Copyright or such successor site as designated by Avaya. The open source software license terms provided as Third Party Terms are consistent with the license rights granted in these Software License Terms, and may contain additional rights benefiting You, such as modification and distribution of the open source software. The Third Party Terms shall take precedence over these Software License Terms, solely with respect to the applicable Third Party Components to the extent that these Software License Terms impose greater restrictions on You than the applicable Third Party Terms.

The following applies only if the H.264 (AVC) codec is distributed with the product. THIS PRODUCT IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD (“AVC VIDEO”) AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

**Service Provider**

THE FOLLOWING APPLIES TO AVAYA CHANNEL PARTNER’S HOSTING OF AVAYA PRODUCTS OR SERVICES. THE PRODUCT OR HOSTED SERVICE MAY USE THIRD PARTY COMPONENTS SUBJECT TO THIRD PARTY TERMS AND REQUIRE A SERVICE PROVIDER TO BE INDEPENDENTLY LICENSED DIRECTLY FROM THE THIRD PARTY SUPPLIER. AN AVAYA CHANNEL PARTNER’S HOSTING OF AVAYA PRODUCTS MUST BE AUTHORIZED IN WRITING BY AVAYA AND IF THOSE HOSTED PRODUCTS USE OR EMBED CERTAIN THIRD PARTY SOFTWARE, INCLUDING BUT NOT LIMITED TO MICROSOFT SOFTWARE OR CODECS, THE AVAYA CHANNEL PARTNER IS REQUIRED TO INDEPENDENTLY OBTAIN ANY APPLICABLE LICENSE AGREEMENTS, AT THE AVAYA CHANNEL PARTNER’S EXPENSE, DIRECTLY FROM THE APPLICABLE THIRD PARTY SUPPLIER.

WITH RESPECT TO CODECS, IF THE AVAYA CHANNEL PARTNER IS HOSTING ANY PRODUCTS THAT USE OR EMBED THE H.264 CODEC OR H.265 CODEC, THE AVAYA CHANNEL PARTNER ACKNOWLEDGES AND AGREES THE AVAYA CHANNEL PARTNER IS RESPONSIBLE FOR ANY AND ALL RELATED FEES AND/OR ROYALTIES. THE H.264 (AVC) CODEC IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO: (I) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD (“AVC VIDEO”) AND/OR (II) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION FOR H.264 (AVC) AND H.265 (HEVC) CODECS MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

**Compliance with Laws**

You acknowledge and agree that it is Your responsibility for complying with any applicable laws and regulations, including, but not

limited to laws and regulations related to call recording, data privacy, intellectual property, trade secret, fraud, and music performance rights, in the country or territory where the Avaya product is used.

**Preventing Toll Fraud**

“Toll Fraud” is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there can be a risk of Toll Fraud associated with your system and that, if Toll Fraud occurs, it can result in substantial additional charges for your telecommunications services.

**Avaya Toll 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. For additional support telephone numbers, see the Avaya Support website: https://support.avaya.com or such successor site as designated by Avaya.

**Security Vulnerabilities**

Information about Avaya’s security support policies can be found in the Security Policies and Support section of https://support.avaya.com/security.

Suspected Avaya product security vulnerabilities are handled per the Avaya Product Security Support Flow (https://support.avaya.com/css/P8/documents/100161515).

**Trademarks**

The trademarks, logos and service marks (“Marks”) displayed in this site, the Documentation, Hosted Service(s), and product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, its licensors, its suppliers, or other third parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation, Hosted Service(s) and product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya Inc.

All non-Avaya trademarks are the property of their respective owners.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

# Contents

# Chapter 1: Introduction

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. It is recommended that the reader of this document has access to the Microsoft Developer Network (MSDN) Library, which provides a complete TAPI reference.

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

**Related links**

Limitations on page 8

## Limitations

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 behavior 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.

**Note:**

If you are using the TAPI interface you may need to increase the size of message receive buffers for the `lineDevStatus` message to allow for the new fields. The required increase in length is 16* (2+ number of User rights groups defined on IP Office).

# Chapter 2: TAPI Link and Wave driver installation

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

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

- Third Party mode
- ACD Queue monitoring
- lineDevSpecific function

To use the Wave functionality, you need to install a Wave licence for each Wave user, in addition to the CTI Link Pro license.

**Related links**


TAPI driver modes on page 9
Enabling single user mode on page 10
Enabling third party mode on page 10
Incoming call queues on page 11
WAV users on page 11
IP Office configuration for TAPI on page 12
Communication loss and recovery on page 12


## TAPI driver modes

TAPI service providers are configured using a Windows Control Panel applet. The name of the applet is not the same across all versions of Windows. The following table indicates the name of the applet and the tab that must be selected within the applet:

| Windows | Control Panel Applet | Tab |
|---|---|---|
| XP Pro | Network and Internet Connections, Phone and Modem Options | Advanced |
| 2000 | Phone and Modem Options | Advanced |

The IP Office TAPI service provider can operate in Single User mode or Third Party mode. You need a license to enable the Third Party mode.

**Related links**

TAPI Link and Wave driver installation on page 9

# Enabling single user mode

### About this task

In Single User mode the TAPI application controls and monitors a single telephony device.

### Procedure

1. Run the appropriate applet for your version of Windows.

   A list of installed TAPI service providers display.

2. Select **Avaya IP Office TAPI Service Provider** and click **Configure**.

   The Avaya TAPI Configuration menu screen displays.

3. In the box labeled “Switch IP Address”, enter the IP address of the IP Office unit.
4. Select the **Single User** option.
5. Enter the user name and password for the extension to be monitored and controlled by TAPI.

   The user name is the name of a person associated with a physical telephone extension.

**Related links**

TAPI Link and Wave driver installation on page 9

# Enabling third party mode

### About this task

In third party mode the TAPI application controls and monitors all telephony devices on a particular IP Office Control Unit. On some versions of Windows, you must reboot the computer or restart the telephony service for configuration changes to take effect.

### Procedure

1. Run the appropriate applet for your version of Windows.

   A list of installed TAPI service providers display.

2. Select **Avaya IP Office TAPI Service Provider** and click **Configure**.

   The Avaya TAPI Configuration menu screen displays.

3. In the box labeled “Switch IP Address”, enter the IP address of the IP Office unit.
4. Select the **Third Party** option.
5. Enter the system password of the IP Office.

   By default, Third Party mode provides 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 controlled by TAPI.

**Related links**

TAPI Link and Wave driver installation on page 9

# Incoming call queues

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

You can select the **ACD Queues** check box, which provides lines to monitor and control the queue of calls against a group.

**Related links**

TAPI Link and Wave driver installation on page 9

# WAV users

A WAV user name begins with “TAPI:”. The IP Office switch attempts to stream audio to WAV users when they are involved in calls.

The audio streaming requires the IP Office Wave driver to be installed on the computer and a Wave driver licence instance for every user. If the Wave driver is not installed, you might still have the **WAV Users** check box selected and still receive WAV user events without a licence.

The TAPI WAV audio stream uses the IP Office data channel taken from the same pool of data channels as voice mail ports. The maximum number of data channels available for simultaneous voice mail and TAPI WAV calls depends on the IP Office Control Unit type.

| Control Unit | Data Channels | Channels usable for Voice mail and TAPI WAV |
|---|---|---|
| Small Office Edition | – | 10 |
| IP403 | 18 | 10 |
| IP406 V1 | 24 | 20 |
| IP406 V2 | 50 | 20 |

*Table continues…*

| Control Unit | Data Channels | Channels usable for Voice mail and TAPI WAV |
|---|---|---|
| IP412 | 108 | 30 |
| IP500 | 48 | 40 |
| IP500 V2 | 48 | 40 |

**Related links**

TAPI Link and Wave driver installation on page 9

# IP Office configuration for TAPI

You can configure the IP Office using IP Office Manager. If the application monitors telephones but does not control the telephones, configuration is not necessary. You can use TAPI with IP Office in the following two ways:

- If the application controls telephones, configure all controlled users as off-hook stations. This causes the phone to return to the idle state when a call is hung up using TAPI. If you do not set this option, the phone remains in a disconnected state until you hang up the phone manually. You can find the **Off-Hook Station** check box on the Telephony tab of the user settings in Manager.
- If you require a special user to handle media, such as an auto attendant, create a new user with a name beginning with “TAPI:”. The WAV users phone number should be in a range that does not conflict with any existing phone numbers or groups.

**Related links**

TAPI Link and Wave driver installation on page 9

# Communication loss and recovery

Avaya recommends that you close all TAPI applications before resetting the switch. The Telephony Service Provider (TSP) closes all open lines and ensures that the switch and all connected TSPs have a consistent state. If the switch is accidentally turned off or a network cable is accidentally unplugged, the TSP detects that the switch is disconnected.

During this time, any calls to TAPI functions that require the TSP to communicate with the switch are rejected. The time 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 might be up to two minutes.

When the TSP detects loss of communication with the switch, an attempt is made to re-establish the connection. After the connection is re-established, the service provider recovers the open lines

and addresses. This is the case even if the loss of communication was caused by the switch rebooting.

The way the communication loss appears to TAPI depends on the version of TAPI used.

- **TAPI 2:** When the TSP loses connection with IP Office, a `lineDevState_OutOfService` message is sent to all open lines. When communication is re-established, a `lineDevState_InService` message is sent to each recovered TAPI line.
- **TAPI 3:** When the TSP loses connection with IP Office, an `ITAddressEvent` message is generated for each address that is registered for such events. These events indicate that the address state is changed to as OutOfService. When the TSP is re-established with IP Office, no events are generated. After communication is re-established, all open TAPI 3 Addresses are recovered.

**Related links**

TAPI Link and Wave driver installation on page 9

# Short code configuration

The following TAPI short code functions are added for IP Office 4.1 and higher:

- Set User Rights Group (short code feature# 196)
- Set User Priority (short code feature# 197)

**Related links**

Configuring the user rights group on page 13
User priority configuration on page 14

## Configuring the user rights group

### About this task

The first character is an integer and not ASCII of integer, that selects the user restrictions group to be changed. The subsequent characters are either:

- The null terminated name of the selected user rights group that you wish to set.
- An empty string to clear user rights group.

### Procedure

1. Set the active user rights group.
2. Set one of the following groups depending on which is currently active:
   a. Working hours user rights group.
   b. Out of hours user rights group.

### Example

```
void TapiLine::SetUserRightsGroup(CString& selstring,
SetURGOption setoption)
```

```
{
int len=4+selstring.GetLength();
TCHAR buffer[100];
buffer[0] = 9;
buffer[1] = 196;
buffer[2]=setoption;
strlcpy(&buffer[3],(LPCTSTR)selstring,16);
HRESULT tr = ::lineDevSpecific(m_hLine,0,NULL,buffer,len);
}
```

# User priority configuration

The following fields are required to configure user priority short code feature# 197:

- The first field is the extension of the user for which the priority is to be set, terminated with a colon (:).
- The second field is a single character representing the integer user priority level from 1 to 5 inclusive.

**Note:**

User priority determines which Alternate Route Selection (ARS) groups a user is permitted to use and levels of outgoing call barring.

### Example

```
void TapiLine::SetUserPriority(TCHAR priority)
```

```
{
TCHAR buffer[100];
TCHAR* p=&buffer[2];
buffer[0] = 9;
buffer[1] = 197;
itoa(m_extension,&buffer[2],10);
int len=strlen(buffer);
p=&buffer[len];
*p=':';
*p++;
*p=priority;
p++;
*p=0;
len=(int)(p-buffer+1);
HRESULT tr = ::lineDevSpecific(m_hLine,0,NULL,buffer,len);
}
```

# Chapter 3: TAPI 2 Reference

**Related links**

TAPI functions on page 15
TAPI 2 Structures on page 32
TAPI events or messages on page 46

## TAPI functions

This section describes each of the TAPI 2.x functions supported by the IP Office TAPI driver, including behaviors or limitations of the functions when used with IP Office.

# lineAddToConference

This function adds the call to the conference.

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

# lineAnswer

This function answers the call that is offered to the application.

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

**Note:**

`UserUserInfo` is not supported and is ignored.

## lineBlindTransfer

This function transfers an active call to a third party. The country code is ignored.

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

## lineCompleteTransfer

This function completes a transfer or the setup of a conference call. Instead of returning a call ID to the conference, this function always returns zero.

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

## lineConfigDialog

This function displays the same TAPI service provider configuration dialog that appears in Control Panel Phone and Modem Options or Telephony. The parameter `lpszDeviceClass` is ignored.

```
LONG
WINAPI
lineConfigDialog(
DWORD dwDeviceID,
HWND hwndOwner,
LPCSTR lpszDeviceClass
);
```

## lineClose

This function closes a line. Use this function when you no longer want to make, receive or monitor calls on a line.

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

# lineDeallocateCall

This function deallocates resources associated with a call. Use this function when the call is in the idle state.

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

# lineDevSpecific

The TSPI allows extended functionality through the `lineDevSpecific` command.

**Note:**

This is only available in the licensed version of the TAPI driver.

The TAPI `lineDevSpecific` function takes a buffer and passes that buffer, unmodified, through to the TSP where it is interpreted as a device-specific function.

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

## Login protocol

To log an ACD agent onto the line being monitored, set the first byte in the buffer to 8. The bytes must be a character string, describing the extension that is logging on. Use this function to log agent 218 onto the line `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
```

## Log off function

Log off can be done by passing the following buffer to the dev specific functions:

```
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 and 6 in the second. The following bytes must be a character string representing the divert destination extension. For example, to set the divert destination to extension 236, use the following function:

```
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 glow to indicate when you have voice mail messages waiting. The number of messages waiting can be controlled by a dev specific command. The IP Office server or other IP Office applications might also control the message waiting 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
```

## Forward or divert settings

This function turns 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 must be three bytes in length and begin with a 9. For example, the following code turns 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

You can only enable and disable membership of the groups to which the user belongs. The user groups are configured in IP Office Manager.

This function enables the user 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);
```

This command disables the user 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);
```

When enabling and disabling group membership, you can choose to disable or enable all group membership by placing a zero in group number. For example `buf[2] = 0`.

**Note:**

The Enable function toggles and is used to both enable and disable membership.

## Intrude

This function intrudes upon another call. The call party to be intruded upon is identified by integer `extnnum`.

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

## Listen

This function enables you to listen to another call. The call party you are listening to is identified with 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 enables you to dial a number on an existing call. Use this function as part of a supervised transfer. The country code is ignored.

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

### Related links

lineSetupTransfer on page 31

## lineDrop

This function hangs up a call. `UserUserInfo` is not supported and is ignored.

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

## lineGenerateDigits

This function generates 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 is 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 generates 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`. Custom tones are not supported, so `dwNumTones` must be set to 0.

```
LONG
WINAPI
lineGenerateTone(
HCALL hCall,
DWORD dwToneMode,
DWORD dwDuration,
DWORD dwNumTones,
LPLINEGENERATETONE const lpTones
);
```

## lineGetAddressCaps

This function retrieves the telephony capabilities of a particular address for a particular line. The capabilities are returned in the `lineAddressCaps` structure.

IP Office lines always have a single address.

```
LONG
WINAPI
lineGetAddressCaps(
HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAddressID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
```

```
LPLINEADDRESSCAPS lpAddressCaps
);
```

**Related links**

lineAddressCaps on page 32

# lineGetAddressID

This function maps a phone number or address assigned to a line device back to its `dwAddressID`. The phone number address range is from zero to the number of addresses minus one returned in the line device capabilities (lineDevCaps). Given that `dwNumAddresses` in `lineDevCaps` is 1, this function always returns 0 in the `DWORD` pointed to by `lpdwAddressID`.

```
LONG
WINAPI
lineGetAddressID(
HLINE hLine,
LPDWORD lpdwAddressID,
DWORD dwAddressMode,
LPCSTR lpsAddress,
DWORD dwSize
);
```

# lineGetAddressStatus

This function enables an application to query the specified address for the current status.

```
LONG
WINAPI
lineGetAddressStatus(
HLINE hLine,
DWORD dwAddressID,
LPLINEADDRESSSTATUS lpAddressStatus
);
```

**Related links**

lineAddressStatus on page 39

# lineGetAppPriority

This function retrieves the priority of your application.

```
LONG
WINAPI
lineGetAppPriority(
LPCSTR lpszAppFilename,
DWORD dwMediaMode,
LPLINEEXTENSIONID lpExtensionID,
DWORD dwRequestMode,
LPVARSTRING lpExtensionName,
LPDWORD lpdwPriority
);
```

# lineGetCallInfo

This function obtains fixed information about a specified call.

```
LONG
WINAPI
lineGetCallInfo(
HCALL hCall,
LPLINECALLINFO lpCallInfo
);
```

### Related links

lineCallInfo on page 40

# lineGetCallStatus

This function retrieves a `lineCallStatus` structure relating to an existing call.

```
LONG
WINAPI
lineGetCallStatus(
HCALL hCall,
LPLINECALLSTATUS lpCallStatus
);
```

### Related links

lineCallStatus on page 42

# lineGetDevCaps

This function retrieves the `lineDevCaps` structure.

```
LONG
WINAPI
lineGetDevCaps(
HLINEAPP hLineApp,
DWORD dwDeviceID,
DWORD dwAPIVersion,
DWORD dwExtVersion,
LPLINEDEVCAPS lpLineDevCaps
);
```

### Related links

lineDevCaps on page 43

# lineGetID

This function gets an 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

This function `lineGetLineDevStatus` returns a device specific buffer.

```
LONG
WINAPI
lineGetLineDevStatus(
HLINE hLine,
LPLINEDEVSTATUS lpLineDevStatus
);
```

**Note:**

IP Office 4.1+ includes a number of additional fields described in the following table. If you are using the TAPI interface you might need to increase the size of message receive buffers for the `lineDevStatus` message to allow for new fields. The required increase in length is 16*(2+ number of user rights groups defined on the IP Office).

The dev specific buffer contains the following information:

| Byte | Contains | Notes |
|---|---|---|
| 0.n | Phone Extension | This line number is monitored as a character string. For example "217". |
| n+1 | 0 | Null terminator for the string. |
| n+2 | Forward on busy | Set the value to 1 if the phone is set to forward on busy. Otherwise set the value to 0. |
| n+3 | Forward on no answer | Set the value to 1 if the phone is set to forward on no answer. Otherwise set the value to 0. |
| n+4 | Forward unconditional | Set the value to 1 if the phone is set to forward all. |
| n+5 | Forward hunt group flag | Set the value to 1 if the phone is set to forward hunt group calls. |
| n+6 | Do Not Disturb | Set the value to 1 if the phone is set to Do Not Disturb. |
| n+7 | Outgoing call bar flag | Set the value to 1 if the phone is barred from making external calls. |
| n+8 | Call waiting on flag | Set the value to 1 if call waiting is enabled for this phone. |
| n+9 | Voice mail on flag | Set the value to 1 if voice mail is enabled for this phone. |
| n+10 | Voice mail ring-back flag | Set the value to 1 if voice mail ring back is enabled for this phone. |
| n+11 | Number of read voice mail messages | The number of read messages. |
| n+12 | Number of unread voice mail messages | The number of voice mail messages waiting for the user. |
| n+13 | Outside call sequence number | Type of ring for external calls. |

*Table continues…*

| Byte | Contains | Notes |
|---|---|---|
| n+14 | Inside call sequence number | Type of ring for internal calls. |
| n+15 | Ring back sequence number | Type of ring for ring back calls. |
| n+16 | No answer time out period | Number of seconds the phone rings before following the no answer action . An example of an action is to forward on no answer and divert to voice mail. |
| n+17 | Wrap up time period | Number of seconds the phone is unable to accept calls after a call. |
| n+18 | Can intrude flag | Set the value to 1 if the phone can intrude upon calls. |
| n+19 | Cannot be intruded upon flag | Set the value to 1 if the phone cannot intruded upon. |
| n+20 | X directory flag | Set the value to 1 if the user does not display in the internal directory. |
| n+21 | Force login flag | The phone is logged out when it starts. The user must log in manually. |
| n+22 | Forced account code flag | Set the value to 1 if the phone is forced to provide a valid account code when making external calls. |
| n+23 | Login code flag | Set the value to 1 if the user has a login code configured. |
| n+24 | System phone flag | Set the value to 1 if this is a system phone. |
| n+25 | Absent message ID | The ID of the absent message. |
| n+26 | Absent message set flag | Set the value to 1 if the absent message with the ID in the previous field is displayed on the phone. |
| n+27 | Voice mail email mode | Set the value to 1 if voice mail email mode is enabled. |
| n+28..m | Extn | The user extension, might be different from the phone extension. |
| m+1 | 0 | Null terminator for Extn string. |
| m+2..p | locale | The locale of the user. |
| p+1 | 0 | Null terminator of locale. |
| p+2..q | Forward destination | The number that the phone is set to divert to. |
| q+1 | 0 | Null terminator for forward destination. |
| 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 | — |

*Table continues…*

| Byte | Contains | Notes |
|---|---|---|
| s+2..t | Do not disturb exception list | A list of numbers that are permitted to call while the phone 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. The Field “t+1” represents the second of these two nulls. If the list is empty, then the data contains a single null which is represented by “t+1”. |
| t+1 | 0 | — |
| t+2..u | Forward on busy number | The number to which calls are redirected when the phone is busy. |
| u+1 | 0 | — |
| u+2 | User's priority | The priority associated with all calls made by this user. |
| u+3 | Group membership | This byte contains the number of groups in which the user is currently enabled. |
| 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 from which the user is currently disabled. |
| 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. |
| Additional fields available for IP Office 4.1+. | | |
| v+8..v | Working Hours User Rights Group Name | Set to blank by default, meaning there are no restrictions to rights. This field enables selection of user rights which might set and lock some user settings. If a Working Hours Time Profile is selected, the Working Hours User Rights are only applied during the times defined by that time profile. Otherwise, they are applied at all times.<br>Maximum length is 15 characters. |
| v+1 | 0 | Null termination for working hours group name. |
| V+2 | Out of Hours User Rights Group Name | Set to blank by default, meaning there are no restrictions to rights. This field enables selection of alternate user rights that are used outside the times defined by the users Working Hours Time Profile.<br>Maximum length is 15 characters. |
| v…w | 0 | Null termination for working hours group name. |

*Table continues…*

| Byte | Contains | Notes |
|---|---|---|
| w+1..x | User Restriction Name List | A list of all User Restriction Group Names defined on the IP Office.<br>Each name is a maximum of 15 characters long and is null terminated.<br>This list enables the TAPI application to present the list of valid values to which the previous two fields can be set. For example in a Combo Box control. |
| x+1 | 0 | Null termination (empty string) of name list. |

# lineHandoff

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

```
LONG
WINAPI
lineHandoff(
HCALL hCall,
LPCSTR lpszFileName,
DWORD dwMediaMode
);
```

# lineHold

This function puts an active call on hold.

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

# lineInitializeEX

This is the first TAPI function that must be used to initialize TAPI. The `lpdwAPIVersion` parameter must be set to at least 0x00020000. This function should be followed by a `lineNegotiateAPIVersion`.

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

# lineMakeCall

This function enables you to make a call.

```
LONG
WINAPI
lineMakeCall(
HLINE hLine,
LPHCALL lphCall,
LPCSTR lpszDestAddress,
DWORD dwCountryCode,
LPLINECALLPARAMS const lpCallParams
);
```

### Related links

lineCallParams on page 42

# lineMonitorDigits

This function enables 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. Detection is done by analyzing 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 `lineDigitMode_DTMFEnd`. Call `lineMonitorDigits` with a `dwDigitMode` of zero to cancel DTMF digit detection.

```
LONG
WINAPI
lineMonitorDigits(
HCALL hCall,
DWORD dwDigitModes
);
```

### Related links

WAV users on page 11

# lineMonitorTone

This function requires the Wave driver involved in the call is 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` is set to NULL to cancel silence detection.

```
LONG
WINAPI
lineMonitorTone(
HCALL hCall,
LPLINEMONITORTONE const lpToneList,
DWORD dwNumEntries
);
```

## lineNegotiateAPIVersion

This function must be called immediately after `lineInitializeEx` to ensure that correct TAPI notifications are sent to your application. Call this function 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` function must be set to `lineMediaMode_InteractiveVoice` for ISDN or 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 you try to open a line that is associated with a Wave user and if there is no Wave User license installed in IP Office, `lineOpen` will return `Lineerr_Resourceunavail`.

## linePark

This function parks a call. Only park mode `lineParkMode_Directed` is supported. The park address can be any alphanumeric string. However, only numeric digits can be entered from a telephone, to restrict your park addresses to numeric strings. The four default park addresses that display in IP Office applications are 1, 2, 3 and 4. Use these numbers to unpark parked calls using the default configuration.

```
LONG
WINAPI
linePark(
HCALL hCall,
DWORD dwParkMode,
LPCSTR lpszDirAddress,
LPVARSTRING lpNonDirAddress
);
```

## lineRedirect

This function redirects the specified offering call to the specified destination address. The country code is ignored.

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

## lineRemoveFromConference

This function removes the call from the conference.

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

## lineSetAppPriority

This function indicates the priority of your application.

```
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 information record.

```
LONG
WINAPI
lineSetAppSpecific(
HCALL hCall,
DWORD dwAppSpecific
);
```

## lineSetCallPrivilege

This function changes your application ownership rights to a particular call.

```
LONG
WINAPI
lineSetCallPrivilege(
HCALL hCall,
```

```
DWORD dwCallPrivilege
);
```

## lineSetStatusMessages

This function enables the application to state which notification messages are required. `dwLineStates` function is set to `lineDevState_All`, and `dwAddressStates` is set to `lineDevState_All`.

```
LONG
WINAPI
lineSetStatusMessages(
HLINE hLine,
DWORD dwLineStates,
DWORD dwAddressStates
);
```

## lineSetupTransfer

This function creates a consultation call in order to perform a supervised transfer. The call that is to be transferred must exist already. The call might be either active or on hold If the call is active, this function puts the call on hold. Call `lineDial` to ring the party to which you are transferring, and call `lineCompleteTransfer` to complete the transfer.

```
LONG
WINAPI
lineSetupTransfer(
HCALL hCall,
LPHCALL lphConsultCall,
LPLINECALLPARAMS const lpCallParams
);
```

## lineShutDown

This function is normally called as your application closes down, enabling you to finish using TAPI line functions.

```
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 or active call is dropped before the held call is retrieved.

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

## lineUnpark

This function retrieves a parked call. `dwAddressID` function must be `0` because IP Office lines only have one address. `lpszDestAddress` function must be the same identifier that was used to park the call.

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

**Related links**

linePark on page 29

# TAPI 2 Structures

**Related links**

TAPI 2 Reference on page 15
lineAddressCaps on page 32
lineAddressStatus on page 39
lineCallInfo on page 40
lineCallParams on page 42
lineCallStatus on page 42
lineDevCaps on page 43

## lineAddressCaps

The `lineAddressCaps` function returns the structure in the following table.

Not all members of this structure are listed.

For more information about `lineAddressCaps`, see the Microsoft TAPI documentation.

| Member | Description/Value |
|---|---|
| dwLineDeviceID | The ID of the line to which the address relates. |
| dwDevSpecificSize | No extra information specific to the device is passed. |
| dwDevSpecificOffset | 0 |
| dwAddressSharing | lineaddressSharing_Private |
| dwAddressStates | 0 |
| dwCallInfoStates | Returns the following possible call info states:<br>• lineCallInfoState_CallID<br>• lineCallInfoState_RelatedCallID<br>• lineCallInfoState_NumOwnerIncr<br>• lineCallInfoState_NumOwnerDecr<br>• lineCallInfoState_NumMonitors<br>• lineCallInfoState_CallerID<br>• lineCallInfoState_CalledID<br>• lineCallInfoState_RedirectionID<br>• lineCallInfoState_RedirectingID<br>• lineCallInfoState_Display<br>• lineCallInfoState_MonitorModes<br>• lineCallInfoState_CallData |
| dwCallerIDFlags | Returns the following possible caller ID flags:<br>• lineCallPartyID_Blocked<br>• lineCallPartyID_OutOfArea<br>• lineCallPartyID_Name<br>• lineCallPartyID_Address<br>• lineCallPartyID_Unknown<br>• lineCallPartyID_Unavail |
| dwCalledIDFlags | Returns the following possible called ID flags:<br>• lineCallPartyID_Blocked<br>• lineCallPartyID_OutOfArea<br>• lineCallPartyID_Name<br>• lineCallPartyID_Address<br>• lineCallPartyID_Unknown<br>• lineCallPartyID_Unavail |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwConnectedIDFlags | Returns the following possible connected ID flags:<br>• lineCallPartyID_Name<br>• lineCallPartyID_Address<br>• lineCallPartyID_Unknown |
| dwRedirectionIDFlags | Returns the following possible redirection ID flags:<br>• lineCallPartyID_Blocked<br>• lineCallPartyID_OutOfArea<br>• lineCallPartyID_Name<br>• lineCallPartyID_Address<br>• lineCallPartyID_Unknown<br>• lineCallPartyID_Unavail |
| dwRedirectingIDFlags | Returns the following possible redirecting ID flags:<br>• lineCallPartyID_Blocked<br>• lineCallPartyID_OutOfArea<br>• lineCallPartyID_Name<br>• lineCallPartyID_Address<br>• lineCallPartyID_Unknown<br>• lineCallPartyID_Unavail |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwCallStates | Returns the following possible call states:<br>• lineCallState_Idle: The call no longer exists.<br>• lineCallState_Offering: A new call has arrived.<br>• lineCallState_Accepted: The call has been claimed by an application.<br>• lineCallState_Dialtone: The caller hears a dial tone.<br>• lineCallState_Dialing: The switch is receiving dialing information.<br>• lineCallState_Ringback: The caller hears ringing.<br>• lineCallState_Busy: The caller hears the busy signal.<br>• lineCallState_Connected: The caller is connected end to end.<br>• lineCallState_Proceeding: Dialing has completed but the call is not yet connected.<br>• lineCallState_Onhold: The call is on hold.<br>• lineCallState_Conferenced: The call is on a conference.<br>• lineCallState_OnHoldPendConf: The call is on hold before conferencing.<br>• lineCallState_OnHoldPendTransfer: The call is on hold before transferring.<br>• lineCallState_Disconnected: The other end has dropped the call.<br>• lineCallState_Unknown: The call state is unknown. |
| dwDialToneModes | Returns the possible dial tone mode lineDialToneMode_Unavail. |
| dwBusyModes | Returns the possible busy mode lineBusyMode_Unavail |
| dwSpecialInfo | Returns the possible special information lineSpecialInfo_Unavail |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwDisconnectModes | Returns the following possible disconnect modes:<br>• lineDisconnectMode_Normal<br>• lineDisconnectMode_Reject<br>• lineDisconnectMode_Pickup<br>• lineDisconnectMode_Forwarded<br>• lineDisconnectMode_Busy<br>• lineDisconnectMode_NoAnswer<br>• lineDisconnectMode_BadAddress<br>• lineDisconnectMode_Unreachable<br>• lineDisconnectMode_Congestion<br>• lineDisconnectMode_Incompatible<br>• lineDisconnectMode_Unavail<br>• lineDisconnectMode_NoDialTone<br>• lineDisconnectMode_QOSUnavail<br>• lineDisconnectMode_Blocked<br>• 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 following possible address cap flags:<br>• lineAddrCapFlags_FwdNumRings<br>• lineAddrCapFlags_Dialed<br>• lineAddrCapFlags_Transferheld<br>• lineAddrCapFlags_Transfermake<br>• lineAddrCapFlags_Conferenceheld<br>• lineAddrCapFlags_Conferencemake<br>• lineAddrCapFlags_FwdStatusValid |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwCallFeatures | Returns the following possible call features:<br>• lineCallFeature_AddToConf<br>• lineCallFeature_Answer<br>• lineCallFeature_BlindTransfer<br>• lineCallFeature_CompleteTransf<br>• lineCallFeature_Dial<br>• lineCallFeature_Drop<br>• lineCallFeature_GenerateDigits<br>• lineCallFeature_Hold<br>• lineCallFeature_Park<br>• lineCallFeature_Redirect<br>• lineCallFeature_RemoveFromConf<br>• lineCallFeature_SetUpTransfer<br>• lineCallFeature_SwapHold<br>• lineCallFeature_Unhold<br>• lineCallFeature_SetCallData |
| dwRemoveFromConfCaps | Returns the possible remove from conference caps lineRemoveFromConf_Any. |
| dwRemoveFromConfState | Returns the possible remove from conference state lineCallState_Onhold. |
| dwTransferModes | Returns the following possible transfer modes:<br>• lineTransferMode_Transfer<br>• lineTransferMode_Conference |
| dwParkModes | Returns the possible park mode lineParkMode_Directed. |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwForwardModes | Returns the following possible forward modes:<br>• lineForwardMode_Uncond<br>• lineForwardMode_UncondExternal<br>• lineForwardMode_UncondSpecific<br>• lineForwardMode_Busy<br>• lineForwardMode_BusyInternal<br>• lineForwardMode_BusyExternal<br>• lineForwardMode_BusySpecific<br>• lineForwardMode_NoAnsw<br>• lineForwardMode_NoAnswInternal<br>• lineForwardMode_NoAnswExternal<br>• lineForwardMode_NoAnswSpecific<br>• lineForwardMode_BusyNa<br>• lineForwardMode_BusyNaInternal<br>• lineForwardMode_BusyNaExternal<br>• lineForwardMode_BusyNaSpecific |
| dwMaxForwardEntries | The maximum number of forwarded entries: 10 |
| dwMaxSpecificEntries | The maximum number of specific entries: 10 |
| dwMinFwdNumRings | The minimum forwarded number of rings: 1 |
| dwMaxFwdNumRings | The maximum forwarded number of rings: 99 |
| dwMaxCallCompletions | 0 |
| dwCallCompletionConds | 0 |
| dwCallCompletionModes | 0 |
| dwNumCompletionMessages | 0 |
| dwCompletionMsgTextEntrySize | 0 |
| dwCompletionMsgTextSize | 0 |
| dwCompletionMsgTextOffset | 0 |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwAddressFeatures | Return the following possible address features:<br>• lineAddrFeature_Forward<br>• lineAddrFeature_MakeCall<br>• lineAddrFeature_SetUpConf<br>• lineAddrFeature_Unpark<br>• lineAddrFeature_Forwardfwd<br>• 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 |

**Related links**

TAPI 2 Structures on page 32

# lineAddressStatus

The lineAddressStatus function returns the structure in the following table.

Not all members of this structure are listed.

For more 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. |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwAddressFeatures | Indicates the following capabilities:<br>• lineAddrFeature_MakeCall<br>• lineAddrFeature_SetUpConf<br>• lineAddrFeature_Unpark |
| dwNumRingsNoAnswer | 5 |
| dwForwardNumEntries | Always 0. |
| dwForwardSize | Always 0. |
| dwForwardOffset | Always 0. |
| dwTerminalModesSize | Always 0. |
| dwTerminalModesOffset | Always 0. |
| dwDevSpecificSize | Always 0. |

**Related links**

TAPI 2 Structures on page 32

# lineCallInfo

The lineCallInfo function returns the is structure in the following table.

Not all members of this structure are listed.

For more information about lineCallInfo, see the Microsoft TAPI documentation.

| Member | Description/Value |
|---|---|
| dwAddressID | Always 0. |
| dwBearerMode | Returns the possible bearer mode lineBearerMode_Voice |
| dwRate | 64000 |
| dwMediaMode | Returns the possible media mode lineMediaMode_InteractiveVoice |
| dwAppSpecific | Set by application. |
| dwCallID | Call ID |
| dwCallParamFlags | Returns the possible call parameter flags lineCallParamFlags_Idle |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwCallStates | Returns the following possible call states:<br>• lineCallState_Idle<br>• lineCallState_Offering<br>• lineCallState_Dialtone<br>• lineCallState_Dialing<br>• lineCallState_Ringback<br>• lineCallState_Busy<br>• lineCallState_Connected<br>• lineCallState_Proceeding<br>• lineCallState_Onhold<br>• lineCallState_Conferenced<br>• lineCallState_OnHoldPendConf<br>• lineCallState_OnHoldPendTransfer<br>• lineCallState_Disconnected<br>• 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 |

*Table continues…*

| Member | Description/Value |
|---|---|
| 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 |

**Related links**

TAPI 2 Structures on page 32

# lineCallParams

The following parameters are recognized in the `lineCallParams` structure that can be passed to `lineMakeCall` and `lineSetupTransfer`.

Not all members of this structure are listed.

For more information on the `lineCallParams`, see the Microsoft TAPI documentation.

| Member | Description/Value |
|---|---|
| dwCallParamFlags | Set this to zero for normal use or enter `lineBearerMode_Voice` to conceal the caller line identifier on the call. |
| dwCalledPartyOffset | Set the called party identifier. |
| dwCallingPartyIDOffset | Set the calling party identifier. |

**Related links**

TAPI 2 Structures on page 32

# lineCallStatus

The `lineCallStatus` function returns the structure in the following table.

Not all members of this structure are listed.

For more information on the `lineCallStatus`, see the Microsoft TAPI documentation.

| Member | Description/Value |
|---|---|
| dwCallState | Returns one of the following states:<br>• lineCallState_Idle: The call no longer exists.<br>• lineCallState_Offering: A new call has arrived.<br>• lineCallState_Accepted: The call is claimed by an application.<br>• lineCallState_Dialtone: The caller hears a dial tone.<br>• lineCallState_Dialing: The switch is receiving dialing information.<br>• lineCallState_Ringback: The caller hears ringing.<br>• lineCallState_Busy: The caller hears the busy signal.<br>• lineCallState_Connected: The call is connected end to end.<br>• lineCallState_Proceeding: Dialing is completed, but the call is not yet connected.<br>• lineCallState_Onhold: The call is on hold.<br>• lineCallState_Conferenced: The call is on a conference.<br>• lineCallState_OnHoldPendConf: The call is on hold before a conference.<br>• lineCallState_OnHoldPendTransfer: The call is on hold before transferring.<br>• lineCallState_Disconnected: The other end has dropped the call.<br>• lineCallState_Unknown: The call state is unknown. |
| dwCallStateMode | Always 0. |
| dwCallPrivilege | The application privilege for this call. |
| dwCallFeatures | The call features available for the call state are indicated by dwCallState. TAPI specifies all possible features, but you can only use those that appear in the dwCallFeatures and lineAddressCaps structures. |
| dwDevSpecificSize | 0 |
| dwDevSpecificOffset | 0 |
| dwCallFeatures2 | 0 |
| tStateEntryTime | Zeros |

**Related links**

TAPI 2 Structures on page 32

# lineDevCaps

The lineDevCaps function returns the structure in the following table.

Not all members of this structure are listed.

For more information on the `lineDevCaps`, see the Microsoft TAPI documentation.

| Member | Description/Value |
|---|---|
| `dwProviderInfoSize` | Indicates the provider name, such as the name of the TSP. |
| `dwSwitchInfoSize` | 0 |
| `dwPermanentLineID` | Unique identifier assigned by Windows. |
| `dwLineNameSize` | Indicates the line name. |
| `dwStringFormat` | Returns the string format `StringFormat_ASCII`. |
| `dwAddressModes` | Returns the address mode `lineAddressMode_AddressID`. |
| `dwNumAddresses` | 1 |
| `dwBearerModes` | Returns the following bearer modes:<br>• `lineBearerMode_Voice`<br>• `lineBearerMode_Speech` |
| `dwMaxRate` | 0 |
| `dwMediaModes` | Returns the media mode `lineMediaMode_InteractiveVoice`. |
| `dwGenerateToneModes` | Returns the generate tone mode `lineToneMode_Beep`. |
| `dwGenerateToneMaxNumFreq` | 0 |
| `dwMonitorToneMaxNumFreq` | 1 |
| `dwMonitorToneMaxNumEntries` | 1 |
| `dwGatherDigitsMinTimeout` | 0 |
| `dwGatherDigitsMaxTimeout` | 0 |
| `dwMedCtlDigitMaxListSize` | 0 |
| `dwMedCtlMediaMaxListSize` | 0 |
| `dwMedCtlToneMaxListSize` | 0 |
| `dwMedCtlCallStateMaxListSize` | 0 |
| `dwDevCapFlags` | Returns the following dev cap flags:<br>• `lineDevCapFlags_CloseDrop`<br>• `lineDevCapFlags_DialBilling`<br>• `lineDevCapFlags_DialQuiet`<br>• `lineDevCapFlags_DialDualTone` |
| `dwMaxNumActiveCalls` | 9 |
| `dwAnswerMode` | Returns the answer mode `lineAnswerMode_None`. |
| `dwRingModes` | 1 |

*Table continues…*

| Member | Description/Value |
|---|---|
| dwLineStates | Returns the following line state:<br>• lineDevState_Ringing<br>• lineDevState_Connected<br>• lineDevState_Disconnected<br>• lineDevState_InService<br>• lineDevState_OutOfService<br>• lineDevState_Open<br>• lineDevState_Close<br>• lineDevState_ReInIt<br>• lineDevState_TranslateChnge<br>• 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 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. |

### Related links

TAPI 2 Structures on page 32

# TAPI events or messages

| Event | Description |
|---|---|
| `line_AppNewCall` | A new call is created. |
| `line_CallInfo` | Information is changed in the `lineCallInfo` structure. |
| `line_CallState` | The state of the call is changed. See `dwCallStates` in the `lineAddressCaps` structure for the list of states supported. |
| `line_lineDevState` | The line device state is changed. The second parameter might be any one of the following:<br>• `lineDevState_DevSpecific`: Dev specific information changed.<br>• `lineDevState_Connected`, `lineDevState_Disconnected`: The connected state of the line changed.<br>• `lineDevState_OutOfService`: The TSP has lost communication with the switch. The line is out of service.<br>• `lineDevState_InService`: The TSP lost connection to the switch but has now recovered and the line is back in service.<br>• `lineDevState_Ringing`: The switch 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 analyze the dev specific buffer for changes. |
| `line_AddressState` | The status of an address is changed on a line that is currently open. |

**Related links**

TAPI 2 Reference on page 15

# Chapter 4: TAPI 3 Reference

**Related links**

[Terminal objects](#) on page 49
[Call hub objects](#) on page 53

## General TAPI objects

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

### ITTAPI interface

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

| Method | Description |
|---|---|
| `Initialize` | Initializes TAPI function.<br>`HRESULT Initialize();` |
| `Shutdown` | Shuts down a TAPI session. Normally called as your applications closes down.<br>`HRESULT Shutdown();` |
| `EnumerateAddresses` | Enumerates the addresses that are currently available.<br>`HRESULT EnumerateAddresses ( IEnumAddress **ppEnumAddress);` |

*Table continues…*

| Method | Description |
|---|---|
| `RegisterCallNotifications` | Sets which new call notifications an application receives. The application must call the method for each address, indicating media types and specifying the requested privileges.<br>`HRESULT RegisterCallNotifications ( ITAddress*pAddress VARIANT_BOOL fMonitor, VARIANT_BOOL fOwner, long lMediaTypes, long lCallbackInstance, long *plRegister );` |
| `put_EventFilter` | Sets the event filter mask.<br>`HRESULT put_EventFilter (long lFilterMask);` |

# Address objects

Address objects represent an entity that can make or receive calls.

## ITAddress

The `ITAddress` interface is the base interface for the Address object.

| Method | Description |
|---|---|
| `get_AddressName` | Displays the name of the address.<br>`HRESULT get_AddressName (BSTR *ppName );` |
| `get_DialableAddress` | Displays the BSTR, which is used to connect to the address.<br>`HRESULT get_DialableAddress ( BSTR*pDialableAddress );` |
| `get_ServiceProviderName` | Displays the name of the Telephony Service Provider (TSP) that supports the address. For example, `Unimdm.tsp` for the Unimodem service provider or `H323.tsp` for the H323 service provider.<br>`HRESULT get_ServiceProviderName ( BSTR*ppName );` |

*Table continues…*

| Method | Description |
|---|---|
| `CreateCall` | Creates a new call object that is used to make an outgoing call and returns a pointer to the objects `ITBasicCallControl` interface.<br>`HRESULT CreateCall ( BSTR*pDialableAddress, Long lAddressType, Long lMediaTypes, ITBasicCallControl**ppCall, );` |

## IEnumAddress

Provides COM-standard enumeration methods for the `ITAddress` interface.

| Method | Description |
|---|---|
| `Next` | Displays the next specified number of elements in the enumeration sequence.<br>`HRESULT Next( ULONG celt, ITAddress **ppElements, ULONG *pceltFetched );` |

## ITMediaSupport

The `ITMediaSupport` interface provides methods that enable an application to discover the media support capabilities for an Address object that displays the interface.

| Method | Description |
|---|---|
| `get_MediaTypes` | Displays the media types supported on the current address.<br>`HRESULT get_MediaTypes ( long *plMediaTypes );` |

# Terminal objects

Terminal objects represent the source or sink of a media stream associated with a call or communication session.

### Related links

TAPI 3 Reference on page 47

# Call objects

The Call objects represent an address connection between the local address and other addresses.

## ITCallInfo

The `ITCallInfo` interface sets a variety of information for a Call object.

| Method | Description |
|---|---|
| `get_Address` | Sets a pointer to the `ITAddress` interface of the Address object.<br>`HRESULT`<br>`get_Address`<br>`(`<br>`ITAddress **ppAddress`<br>`);` |
| `get_CallState` | Sets a pointer to the current call state. For example, CS_IDLE.<br>`HRESULT`<br>`get_CallState (`<br>`CALL_STATE *pCallState`<br>`);` |
| `get_CallInfoString` | Gets call information items described by a string. For example, displayable address.<br>`HRESULT`<br>`get_CallInfoString`<br>`(`<br>`CALLINFO_STRING CallInfoString,`<br>`BSTR *ppCallInfoString`<br>`);` |
| `SetCallInfoBuffer` | By default, TAPI 3.0 (Windows 2000) only allows this function on a call that is in the Idle state. In TAPI 3.1 (Windows XP), this function allows call data to be set on calls in the connected state by passing `CIB_CallDataBuffer` as the `CallInfoBuffer` parameter.<br>`HRESULT`<br>`SetCallInfoBuffer (`<br>`CALLINFO_BUFFER CallInfoBuffer,`<br>`DWORD dwSize,`<br>`BYTE* pCallInfoBuffer`<br>`);` |

## ITBasicCallControl

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

| Method | Description |
|---|---|
| `Connect` | Attempts to complete the connection of an outgoing call.<br>`HRESULT`<br>`Connect(`<br>`VARIANT_BOOL fSync`<br>`);` |
| `Answer` | Answers an incoming call. This method can succeed only if the call state is `CS_Offering`.<br>`HRESULT`<br>`Answer();` |
| `Disconnect` | Disconnects the call. The call state transits to `CS_Disconnected` after the method completes successfully.<br>`HRESULT`<br>`Disconnect(`<br>`DISCONNECT_CODE code`<br>`);` |
| `Hold` | Places or removes the call from the hold.<br>`HRESULT`<br>`Hold(`<br>`VARIANT_BOOL fHold`<br>`);` |
| `SwapHold` | Swaps the active call with the specified call on hold.<br>`HRESULT`<br>`SwapHold(`<br>`ITBasicCallControl *pCall`<br>`);` |
| `ParkDirect` | Parks the call at a specified address.<br>`HRESULT`<br>`ParkDirect(`<br>`BSTR pParkAddress`<br>`);` |
| `Unpark` | Gets the call from park.<br>`HRESULT`<br>`Unpark()`<br>`;` |
| `BlindTransfer` | Performs a blind or single-step transfer of the specified call to the specified destination address.<br>`HRESULT BlindTransfer(`<br>`BSTR pDestAddress`<br>`);` |
| `Transfer` | Transfers the current call to the destination address.<br>`HRESULT Transfer(`<br>`ITBasicCallControl*pCall,`<br>`VARIANT_BOOL fSync`<br>`);` |

*Table continues…*

| Method | Description |
|---|---|
| `Finish` | Finishes a conference or a transfer on a consultation call.<br>`HRESULT Finish(`<br>`FINISH_MODE finishMode`<br>`);` |
| `Conference` | Adds a consultation call to the conference in which the current call is a participant.<br>`HRESULT Conference(`<br>`ITBasicCallControl *pCall,`<br>`VARIANT_BOOL fSync`<br>`);` |
| `RemoveFromConference` | Removes the call from a conference if involved in one.<br>`HRESULT RemoveFromConference();` |

# ITCallStateEvent

The `ITCallStateEvent` interface provides methods that retrieves the description of call state events.

| Method | Description |
|---|---|
| `get_Cause` | Gets the cause associated with the event.<br>`HRESULT`<br>`get_Cause (`<br>`CALL_STATE_EVENT_CAUSE *pCEC`<br>`);` |
| `get_State` | Gets information on the new call state.<br>`HRESULT`<br>`get_State (`<br>`CALL_STATE *pCallState`<br>`);` |
| `get_Call` | Gets a pointer to the call information interface for the call on which the event has occurred.<br>`HRESULT`<br>`get_Call`<br>`ITCallInfo **ppCallInfo`<br>`);` |

# ITCallNotificationEvent

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

| Method | Description |
|---|---|
| `get_Call` | Returns the `ITCallInfo` interface on which a call event is occurred.<br>`HRESULT`<br>`get_Call`<br>`ITCallInfo **ppCall`<br>`);` |

## ITCallInfoChangeEvent

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

| Method | Description |
|---|---|
| `get_Call` | Returns the `ITCallInfo` interface on which call information is changed.<br>`HRESULT`<br>`get_Call`<br>`ITCallInfo **ppCall`<br>`);` |

# Call hub objects

The Call Hub object method retrieves participant information in a multi party call. IP Office does not support call hubs. Call hub events might be received and ignored.

**Related links**

TAPI 3 Reference on page 47

# Chapter 5: TAPI 3 Enumerated Types

**Related links**


Call_State on page 54
CallInfo_String on page 55
Disconnect_Code on page 55
Call_Statr_Event_Cause on page 56


## Call_State

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

| Member | Value | Description |
|---|---|---|
| `CS_IDLE` | 0 | A call is created, but Connect is not called yet. A call can never transit into the idle state. |
| `CS_INPROGRESS` | 1 | Connect is called, and the service provider is working on making a connection. This state is valid only for outgoing calls. This message is optional, because a service provider might have a call transition directly to the connected state. |
| `CS_CONNECTED` | 2 | Call is connected to the remote end and communication can take place. |
| `CS_DISCONNECTED` | 3 | Call is disconnected. This can occur for several reasons outlined in Disconnect_code. |
| `CS_OFFERING` | 4 | A new call appears, and is offered to an application. If the application has owner privileges, the call is either answered or disconnected while the call is in the offering state. |
| `CS_HOLD` | 5 | The call is on hold state. |
| `CS_QUEUED` | 6 | The call is in the queue. |

**Related links**


TAPI 3 Enumerated Types on page 54
Disconnect_Code on page 55

# CallInfo_String

The `CallInfo_String` enum is used by `ITCallInfo` methods that sets and gets 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 is redirected. |
| `CIS_REDIRECTIONIDNUMBER` | 7 | The number of the location to which a call is 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 friendly name of the called party. |
| `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. |

### Related links

[TAPI 3 Enumerated Types](#) on page 54

# Disconnect_Code

The `Disconnect_Code` enum is used by the `ITBasicCallControl::Disconnect` method.

| Member | Value | Description |
|---|---|---|
| `DC_NORMAL` | 0 | The call is disconnected as part of the normal cycle of the call. |
| `DC_NOANSWER` | 1 | The call was not answered. An application might set a certain amount of time for the user to answer the call. If the user does not answer, the application can disconnect the call with the `NOANSWER` code. |
| `DC_REJECTED` | 2 | The user rejected the call. |

**Related links**

TAPI 3 Enumerated Types on page 54

# Call_Statr_Event_Cause

The `Call_Statr_Event_Cause` enum is returned by the`ITCallStateEvent::get_Cause` method.

| Member | Value | Description |
|---|---|---|
| `CEC_NONE` | 0 | The call event did not occur. |
| `CEC_DISCONNECT_NORMAL` | 1 | The call ended and disconnected normally. |
| `CEC_DISCONNECT_BUSY` | 2 | The outgoing call failed to connect because the remote end is busy. |
| `CEC_DISCONNECT_BADADDRESS` | 3 | The outgoing call failed because the destination address is incorrect. |
| `CEC_DISCONNECT_NOANSWER` | 4 | The outgoing call failed because the remote end did not answer. |
| `CEC_DISCONNECT_CANCELLED` | 5 | The outgoing call failed because the caller disconnected the call. |
| `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. |

**Related links**

TAPI 3 Enumerated Types on page 54

# Chapter 6: IP Office media service provider

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.

## Media service provider usage

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 can 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.

## Device specific interfaces

The device specific interfaces are implemented on the Address and Call objects of the MSP. TAPI 3.0 delegates queries for interfaces it does not recognize to the MSP. For example, if you have a pointer to an `ITAddress` interface, you can call Query Interface to retrieve a pointer to the `ITDivert` interface. The following code is an example from the DevSpice sample:

```
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`

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`.

# ITACDAgent interface

| Name | Description |
|---|---|
| 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 the line. The user must have **Force Logon** set in IP Office Manager. |
| Login(BSTR extn) | Logs the user on to 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 IP Office 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 IP Office Manager. |
| SetAccountCode(BSTR extn) | Sets the account code for the current call. |

# ITGroup interface

The `ITGroup` interface contains functions to take the user in and out of groups and to intercept calls that present themselves at other phones in the group.

| Name | Description |
|---|---|
| PickupAny(void) | Equivalent to executing the `CallPickupAny shortcode` on the user's terminal. For more information, see IP Office Manager. |
| 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. |

*Table continues…*

| Name | Description |
|---|---|
| Enable(BSTR groupextn) | Enables the user's membership of the given group. If groupextn is an empty string, the user is enabled in all member groups. |
| Disable(BSTR groupextn) | Disables the user's membership of the given group. If groupextn is an empty string, the user is disabled in all member groups. |

# ITDivert interface

The `ITDivert` interface contains functions for getting and setting the divert flags for the address.

| Name | Description |
|---|---|
| GetDivertAllDestination(BSTR* pDestination) | Gets the current *Divert All* destination and returns the result in the *pDestination* value. |
| SetDivertAllDestination(BSTR dest) | — |
| GetDivertSettings(DWORD* pdwDivertSets) | 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. |
| SetForwardAll(VARIANT_BOOL bOn) | Toggles the ForwardAll setting for the user. |
| SetForwardBusy(VARIANT_BOOL bOn) | Toggles the ForwardBusy setting for the user. |
| SetForwardNoAnswer(VARIANT_BOOL bOn) | Toggles the ForwardNoAnswer setting for the user. |
| SetDoNotDisturb(VARIANT_BOOL bOn) | Toggles the DoNotDisturb setting for the 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`, but only by the `Fire_DivertSettingsChanged` function on the `IPOfficePrivateEvents` interface.

# ITPlay interface

The `ITPlay` is implemented on the MSP Call object for recording and playing Wave files.

| Name | Description |
|---|---|
| StartPlay(BSTR FileName) | `FileName` must be a complete path to the Wave file to play. |
| StopPlay() | — |
| StartRecord(BSTR FileName) | `FileName` must be a complete path to the Wave file to record to. |
| StopRecord() | — |

Playing and recording can be stopped and started at any time on the call. Recording is used only as single file per call though, and appended to the file if recording is stopped and restarted. you must not 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 interface

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

| Name | When used |
|---|---|
| OnUserLogin(void) | When an agent or a user with **Force Logon** set in IP Office Manager logs in. |
| OnUserLogout(void) | When an agent logs out. |
| OnDivertSettingsChanged(DWORD dwDivertSettings) | When the user changes one of their divert setting flags, such as *Do Not Disturb*, *Divert On Busy*, or the *Divert All destination*. The bits in `dwDivertSettings` are set using the `IP_OFFICE_DIVERT_SETTINGS` enum. |
| OnGroupChanged(DWORD dwGroupCount) | 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) | When the number of waiting voice mail messages changes. The new value is given in `dwNumMessages`. |

# Media streaming capabilities

The IP Office MSP handles media streaming to any Wave user. You must select terminals onto the streams that the MSP displays for a call. For more information about media streaming, see MSDN and an example in the DevSpice sample.

**Note:**

IP Office streams are bi-directional, and there is only a single stream per call. Both capture and render terminals are accepted on the same stream, but only one of each.

The MSP contains the functionality of the IP Office Wave driver. The IP Office Wave driver must be installed on each machine that provides media streaming to Wave users.

**Note:**

If you want to only monitor Wave users using TAPI, do not install Wave driver. Otherwise, Wave licence instances are consumed for every Wave user line that you open.

# Chapter 7: Additional Help and Documentation

The following pages provide sources for additional help.

**Related links**


Additional Manuals and User Guides on page 62
Getting Help on page 62
Finding an Avaya Business Partner on page 63
Additional IP Office resources on page 63
Training on page 64


## Additional Manuals and User Guides

The Avaya Documentation Center website contains user guides and manuals for Avaya products including IP Office.

- For a listing of the current IP Office manuals and user guides, look at the Avaya IP Office™ Platform Manuals and User Guides document.
- The Avaya IP Office Knowledgebase and Avaya Support websites also provide access to the IP Office technical manuals and users guides.
  - Note that where possible these sites redirect users to the version of the document hosted by the Avaya Documentation Center.

For other types of documents and other resources, visit the various Avaya websites (see Additional IP Office resources on page 63).

**Related links**


Additional Help and Documentation on page 62


## Getting Help

Avaya sells IP Office through accredited business partners. Those business partners provide direct support to their customers and can escalate issues to Avaya when necessary.

If your IP Office system currently does not have an Avaya business partner providing support and maintenance for it, you can use the Avaya Partner Locator tool to find a business partner. See Finding an Avaya Business Partner on page 63.

**Related links**

Additional Help and Documentation on page 62

# Finding an Avaya Business Partner

If your IP Office system currently does not have an Avaya business partner providing support and maintenance for it, you can use the Avaya Partner Locator tool to find a business partner.

### Procedure

1. Using a browser, go to the Avaya Website at https://www.avaya.com
2. Select **Partners** and then **Find a Partner**.
3. Enter your location information.
4. For IP Office business partners, using the **Filter**, select **Small/Medium Business**.

**Related links**

Additional Help and Documentation on page 62

# Additional IP Office resources

In addition to the documentation website (see Additional Manuals and User Guides on page 62), there are a range of website that provide information about Avaya products and services including IP Office.

- **Avaya Website** (https://www.avaya.com)

  This is the official Avaya website. The front page also provides access to individual Avaya websites for different regions and countries.

- **Avaya Sales & Partner Portal** *(*https://sales.avaya.com*)*

  This is the official website for all Avaya business partners. The site requires registration for a user name and password. Once accessed, the portal can be customized for specific products and information types that you wish to see and be notified about by email.

- **Avaya IP Office Knowledgebase** *(*https://ipofficekb.avaya.com*)*

  This site provides access to an online, regularly updated version of IP Office user guides and technical manual.

- **Avaya Support** *(*https://support.avaya.com*)*

This site provide access to Avaya product software, documentation and other services for Avaya product installers and maintainers.

- **Avaya Support Forums** *(*https://support.avaya.com/forums/index.php*)*

  This site provides a number of forums for discussing issues.

- **International Avaya User Group** *(*https://www.iuag.org*)*

  This is the organization for Avaya customers. It provides discussion groups and forums.

- **Avaya DevConnect** *(*https://www.devconnectprogram.com/*)*

  This site provides details on APIs and SDKs for Avaya products, including IP Office. The site also provides application notes for 3rd-party non-Avaya products that interoperate with IP Office using those APIs and SDKs.

- **Avaya Learning** *(*https://www.avaya-learning.com/*)*

  This site provides access to training courses and accreditation programs for Avaya products.

**Related links**

Additional Help and Documentation on page 62

# Training

Avaya training and credentials are designed to ensure our Business Partners have the capabilities and skills to successfully sell, implement, and support Avaya solutions and exceed customer expectations. The following credentials are available:

- Avaya Certified Sales Specialist (APSS)
- Avaya Implementation Professional Specialist (AIPS)
- Avaya Certified Support Specialist (ACSS)

Credential maps are available on the Avaya Learning website.

**Related links**

Additional Help and Documentation on page 62

# Index

## R



## S



## T



## U



## W