TSP++ Version 3.0

TSP++ Programmers Reference Guide

Building 3rd Party TAPI Service Providers for Windows NT 4.0, Windows 2000 and Windows XP

TSP+++ Version 3.046

Table of Contents

CServiceProvider

In the TSP++ class library, the basic class that will always be present is a class derived from **CServiceProvider**. The **CServiceProvider** object is the application object that controls the initialization of the TSP itself.

When the TSP is being initialized by TAPISRV.EXE, it will call the **TSPI_providerEnumDevices** and **TSPI_lineProviderInit** functions. Since no other objects exist at that moment in time, the calls are mapped to **CServiceProvider** methods that process the initialization of the TSP.

Service Provider Initialization

All line and telephone device objects are created during initialization of the service provider. This occurs when TAPI calls the **TSPI_providerInit** function.

Once the **CServiceProvider::providerInit** method completes, all the main telephony objects should be initialized and ready for use. The derived class could then configure the lines/phones in the appropriate fashion, overriding the default values specified by the library if necessary.

Persistent data storage

Any persistent data that needs to be stored by the service provider should use the following methods in the library, which have been created for this purpose:

- **ReadProfileString**
- **ReadProfileDWord**
- **WriteProfileString,**
- **WriteProfileDWord**

These methods store the information into an area dedicated to the provider. This area will be a section of the registry under

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Telephony

Required and Typical Overrides

The only method that generally must be overridden in this object is the constructor. This is used to initialize the name of the service provider (which allows the above listed profile methods to find data associated with the service provider). The constructor is also where any replacement of object types needs to take place (using

SetRuntimeObjects), for more information on this topic, see the section titled *Class Overriding* in the user's guide.

If the service provider is not using the built in UI object types (see the related document concerning the user interface library) to store information in the registry, then the device count will not be available for TAPI and the **providerEnumDevices** method must be overridden. It should fill in the count of lines and phones supported by the provider as this information is used by TAPI to call the **providerInit** method after **providerEnumDevices** returns.

Constructor and Destructor

CServiceProvider	Constructs a **CServiceProvider** object. This takes the TSP name, user-interface DLL name, the highest version of TAPI to negotiate to, and provider information.
~CServiceProvider	Destructor that deallocates all the global information in the service provider object.

Initialization - Protected Members

SetRuntimeObjects	Override the default object types for each of the basic objects (line, phone, device, request, class, and address).

Operations - Public Members

AddTimedCall	This adds the specified call to an internal list of calls that receive a periodic timer from the TSP++ library.
DeleteProfile	Delete a profile from the registry or Telephon.INI
GetConnInfoFromLineDeviceID	Return a pointer to a **CTSPILineConnection** based on a line device id.
GetConnInfoFromPhoneDeviceID	Return a pointer to a **CTSPIPhoneConnection** based on a phone device id.
GetCurrentLocation	Returns the current location.
GetDevice	Return a pointer to a **CTSPIDevice** based on a device id.
GetDeviceCount	Returns a count of existing devices.
GetDeviceByIndex	Returns a pointer to a **CTSPIDevice** based on an index.
GetProviderInfo	Returns provider specific information that was supplied by the constructor.
GetResourceInstance	Returns the HINSTANCE associated with resources for the TSP.
GetShutdownEvent	This returns the Win32 event object that is set

	when the provider is being shutdown. It may be used to signal any worker threads to exit.
GetSupportedVersion	Returns the version of TAPI the service provider negotiates at. This would be the highest level of TAPI supported by the service provider. The library supports all versions (1.3-2.1)
GetSystemVersion	Returns the version of TAPI.DLL that started the service provider.
GetUIManager	This returns the name of the user-interface DLL to load when TAPI needs to display a dialog. The name is set on the constructor of the service provider object.
IsRunningUnderTapiSrv	This returns whether the TSP is running under TAPISRV.EXE (or DLL) vs. being run from the control panel applet.
ReadProfileDWord	Reads a DWORD from the registry section for this provider.
ReadProfileString	Reads a string from the registry section for this provider.
RemoveTimedCall	This removes the specified call from the timed call list.
RenameProfile	Rename a profile in the registry section for this provider.
WriteProfileDWord	Writes a DWORD to the registry section for this provider.
WriteProfileString	Writes a string to the registry section for this provider.

Operations – Protected Members

CanHandleRequest	Return whether this service provider can handle a type of request.

Overridables - Public Members

CheckDialableNumber	Determine if an address is valid. Splits the dialable address into component **DIALINFO** objects that contain address, name, and sub-address information. This is called for any dialable address passed into the service provider.
ConvertDialableToCaononical	Converts a dialable number to the best canonical equivalent that can be determined based on the selected location information. This is called to format caller id information.
DeleteCall	This is called to move a call to the delete list where

	it will eventually be deallocated by the TSP.
DetermineAreaCode	This method is used to determine the proper area code for a dialable number.
DetermineCountryCode	This method is used to determine the proper country code for a dialable number.
GetDialableNumber	Strips out any non-specified characters from a dialable string. Always leaves any numbers in place.
MatchTones	Match a set of frequencies for tone monitoring. The default implementation compares each frequency for an exact match.
ProcessCallParameters	Check a set of call parameters for validity. This should be overridden if special requirements are needed for **LINECALLPARAM** structures passed from TAPI.
TraceOut	This is called to place a debug buffer into the persistent debug log.

Overridables - TAPI Members

providerCreateLineDevice	This is called in response to the TSPI_providerCreateLineDevice command. The default implementation assigns the correct line device ID to the line object.
providerCreatePhoneDevice	This is called in response to the TSPI_providerCreatePhoneDevice command. The default implementation assigns the correct line device ID to the phone object.
providerEnumDevices	This is called in response to the TSPI_providerEnumDevices command. The default implementation returns the count of lines and phones listed in the registry.
providerFreeDialogInstance	This is called when a user-interface dialog is being terminated. It frees the data associated with the dialog.
providerGenericDialogData	This is called to provide information to an existing generic user-interface dialog.
providerInit	This is called in response to the TSPI_providerInit command. The default implementation initializes the device object specified by the provider id.
providerShutdown	This is called in response to the TSPI_providerShutdown command. It is one of the final methods called before the TSP is unloaded by TAPISRV.
providerUIIdentify	Identifies the user-interface DLL that is loaded

	into the application process space for UI events.
lineDevSpecific	This is called in response to the TSPI_lineDevSpecific command. The default implementation calls the **CTSPILineConnection::DevSpecificFeature** method.
lineForward	This is called in response to the TSPI_lineForward command. The default implementation validates and copies parameters and calls the **CTSPIAddressInfo::Forward** method.
lineNegotiateTSPIVersion	This is called in response to the TSPI_lineNegotiateTSPIVersion command. The default implementation performs a negotiation that takes into account the highest version the provider indicated, and down to TAPI 1.3.
lineSetCurrentLocation	This sets the current location identifier for the TSP. It is called in response to a TSPI_lineSetCurrentLocation command from TAPI. The default implementation reloads its location information for caller-id display.
lineSetMediaControl	This is called in response to the TSPI_lineSetMediaControl command. The default implementation validates and copies parameters and calls either the **CTSPICallAppearance, CTSPIAddressInfo,** or **CTSPILineConnection::SetMediaControl** method depending on the parameters passed.

CServiceProvider::AddTimedCall

void AddTimedCall(CTSPICallAppearance* *pCall***);**

pCall	The call object to add to the internal array.

Remarks

This method is used to add a call object to the internal array of calls that *need* timer events. This will cause the call's **OnTimer** method to be called every second so that it may process information. The timer will continue to run through this call until the call is removed using the **RemoveTimedCall** method.

This is used internally by TSP++ to manage digit, tone, and media gathering events that have timeout conditions associated with them. It may be used by the derived provider in order to do some other form of work on the call by overriding the **CTSPICallAppearance::OnTimer** method.

CServiceProvider::CanHandleRequest

Protected

bool CanHandleRequest (DWORD *dwFunction***)**

dwFunction	The TSPI entry-point to verify.

Remarks

This method is used to determine if an asynchronous request packet should be generated for each available event type. This is determined based on whether the function is exported by the TSP.

Return Value

TRUE	if the service provider supports the request type
FALSE	if the request cannot be processed and should return an error.

CServiceProvider::CheckDialableNumber

virtual LONG CheckDialableNumber(CTSPILineConnection* *pLine***,**
CTSPIAddressInfo* *pAddr***, LPCTSTR** *lpszDigits***,**
TDialStringArray* *parrEntries***, DWORD** *dwCountry***,**
LPCTSTR *pszValidChars***=NULL);**

pLine	Line object the dial string is associated with
pAddr	Address object the dial string is associated with
lpszDigits	Digits to dial
parrEntries	Returning array of **DIALINFO** structures
dwCountry	Country code to use with dialing code.
pszValidChars	Valid list of characters in a dial string. NULL indicates default TAPI restrictions.

Remarks

This method checks the passed dialable number to determine if we support the dialable address. The final form address is returned in the as a series of **DIALINFO** structures (at least one) for each valid address found in the dialable string.

The valid characters for a dial string are: **0123456789ABCD*#!WPT@$+,**.

If the hardware has additional valid characters, then the final parameter needs to include the above list along with the additional characters to pass through the dial string.

Any character found in the dial string that is not in the valid list is stripped from the resultant **DIALINFO** structure.

For more information on dial strings, see the section on *Dialing Information* or the section on *Structures: DIALINFO*.

Return Value

TAPI result code or FALSE for success.

CServiceProvider::ConvertDialableToCanonical

virtual TString ConvertDialableToCanonical (LPCTSTR *pszNumber***,**
DWORD *dwCountry***=0, bool** *fInbound***=false)**

pszNumber	Dialable number.
dwCountry	Country code to use.
FInbound	True for an inbound call

Remarks

This method converts a dialable number to a standardized canonical format. The standard canonical format is: **+1 (800) 555-1212**. This is the format all phone numbers are reported back to TAPI in (for things such as caller id).

For more information on dial strings, see the section on *Dialing Information* or the section on *Structures: DIALINFO*.

Return Value

String with number formatted in standard canonical format.

CServiceProvider::CServiceProvider

CServiceProvider(LPCTSTR *pszAppName***, LPCTSTR** *pszProviderInfo***,**
DWORD *dwTapiVer* **= TAPIVER_21);**

pszAppName	A null-terminated string that contains the name of the user-interface DLL that TAPI should load for all user-interface events.
pszProviderInfo	A null-terminated string with the copyright string for the provider that is reported as part of the provider information and capabilities structure.
dwTapiVer	This is the minimum negotiable version of TAPI to allow the provider to run under. This will allow TAPI to determine if the provider is too new to run on the

machine. Under TAPI 2.x, this value must be at least **TAPIVER_20**.

Remarks

Constructs a **CServiceProvider** object. Only one object should be created within a service provider, and it should be a global variable within the provider source code.

CServiceProvider::DeleteCall

virtual void DeleteCall(CTSPICallAppearance* *pCall***);**

pCall	The call object to move to the deleted array

Remarks

This method is used to move calls to the provider *deleted* array. It is automatically invoked when the call is deallocated by TAPISRV.

Calls are kept in a deleted array for up to 30 seconds in order to ensure that TAPI does not send a bad call pointer back into the library. If the number of calls in the deleted array goes up, the thread responsible for really deleting the call will adjust the timeout values automatically.

This method should never be called directly. It is documented so that it may be overridden if desired.

CServiceProvider::DeleteProfile

bool DeleteProfile (DWORD *dwPPid***);**

dwPPid	Provider ID assigned by TAPI to the device storing data.

Remarks

This method destroys the persistent data associated with the service provider and device id from the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

This method is automatically called when the provider is de-installed using **TSPI_providerRemove**.

Return Value

TRUE if the section was removed.

CServiceProvider::DetermineAreaCode

virtual TString DetermineAreaCode(const TString& *strCountryCode***,**
TString& *strInput***);**

strCountryCode	Country code for this number
strInput	Dialable number input

Remarks

This function is used to locate the area code information within the given dialing string. It is coded specifically for the U.S. and requires override for many countries. In some cases, the area code cannot be fully determined (in Europe for instance where it is a variable-number of digits). Override this to do what you need if it is not sufficient for your locale.

Return Value

Returning area code.

CServiceProvider::DetermineCountryCode

virtual TString DetermineCountryCode(const TString& *strInput***) const;**

strInput Dialable number input

Remarks

This function is used to locate the country code information within the given dialing string. If the given country is not known then it simply returns the passed input. Override this to do what you need if it is not sufficient for your locale.

Return Value

Returning country code portion of the string.

CServiceProvider::GetConnInfoFromLineDeviceID

CTSPILineConnection* GetConnInfoFromLineDeviceID(DWORD *dwDevId***);**

dwDevID TAPI Device ID to search for.

Remarks

This method searches the line connection array for a **CTSPILineConnection** object that matches the passed device identifier. If no object is found with the matching ID, NULL is returned.

Return Value

Line object that is associated to the passed TAPI line device id. If no object is found this method returns NULL.

CServiceProvider::GetConnInfoFromPhoneDeviceID

CTSPIPhoneConnection* GetConnInfoFromPhoneDeviceID(
DWORD *dwDeviceID***);**

dwDeviceID TAPI Device ID to search for.

Remarks

This method searches the phone connection array for a **CTSPIPhoneConnection** object that matches the passed device identifier. If no object is found with the matching ID, NULL is returned.

Return Value

Phone object that is associated to the passed TAPI phone device id. If no object is found this method returns NULL.

CServiceProvider::GetCurrentLocation

DWORD GetCurrentLocation() const;

Remarks

This method simply returns the current location information that has been set by TAPI.

Return Value

The current TAPI location identifier.

CServiceProvider::GetDevice

CTSPIDevice* GetDevice(DWORD *dwProviderID***) const;**

dwProviderID	Provider ID to locate. This is a sequential index from within TAPI that might not start with zero.

Remarks

This method searches the device array for a **CTSPIDevice** object that matches the passed provider identifier. If no object is found with the matching ID, NULL is returned.

TAPI assigns provider identifiers in a sequential fashion, starting at zero and moving through each service provider installed in the system. Each device in the library will be assigned a unique identifier. Since devices can be removed from within TAPI, the provider identifiers might not be complete (i.e. they can have holes in the numeric sequence). For this reason, do not assume any order or sequence to the identifiers.

If you want to locate a device based on a zero-based index within the provider, use the method **CServiceProvider::GetDeviceByIndex**.

Return Value

The device object that is associated to the passed TAPI provider id. If no object is found this method returns NULL.

CServiceProvider::GetDeviceByIndex

CTSPIDevice* GetDeviceByIndex(int *iDevice***) const;**

iDevice	Zero-based index to locate. This is a sequential index from within the service provider.

Remarks

This method returns the device at the specified position within the device array. If there is no device at that position, NULL is returned.

If you want to locate a device based on a provider id from TAPI, use the method **CServiceProvider::GetDevice**.

Return Value

Device object that is associated to the passed index. If no object is found this method returns NULL.

CServiceProvider::GetDeviceCount

DWORD GetDeviceCount() const;

Remarks

This method returns the count of devices present in the service provider. There should always be at least once device as long as the provider has been completely initialized.

Return Value

Count of devices installed in TAPI that are being run out of this service provider.

CServiceProvider::GetDialableNumber

virtual TString GetDialableNumber (LPCTSTR *pszNumber*,
LPCTSTR *pszAllowChar* **= NULL) const;**

pszNumber	The number to dial.
pszAllowChar	List of allowable characters for a dialable string.

Remarks

This method strips information out of a phone number that is not required to dial the number. This includes things such as area-code braces, pause codes, wait-for-dialtone requests, etc.

The allowable character set allows the caller to determine what is left when the number is returned.

Return Value

String with number formatted in standard dialable format.

CServiceProvider::GetProviderInfo

LPCTSTR GetProviderInfo() const;

Remarks

This method returns the information about the provider, that was supplied, on the constructor.

Return Value

NULL terminated pointer to the constant string that was passed to the constructor of the provider.

CServiceProvider::GetResourceInstance

HINSTANCE GetResourceInstance() const;

Remarks

This method returns the handle to the resources located within the service provider. A developer may use this returned handled to locate icons, string tables, etc. that are embedded within the resource table of the service provider.

Return Value

Windows handle to the resource table for use with **LoadResource** or any other resource-oriented method.

CServiceProvider::GetShutdownEvent

HANDLE GetShutdownEvent() const

Remarks

This method returns the Win32 event object that was created when the service provider was loaded. This event stays in the non-signaled state until the provider shutdown process starts, at which point the TSP++ library signals the event and releases any threads blocked on it.

This event may be used in worker threads to get them to exit when the provider shuts down.

Do not manipulate this event in any way during the lifetime of the service provider – unpredictable results may happen.

Return Value

Win32 event object that will be signaled when the service provider shuts down.

CServiceProvider::GetSupportedVersion

DWORD GetSupportedVersion() const;

Remarks

This method returns the version of TAPI that the service provider is willing to negotiate to. This will be the highest level of TAPI that is supported by this provider.

Return Value

The TAPI version that was supplied to the constructor of the object

CServiceProvider::GetSystemVersion

DWORD GetSystemVersion() const;

Remarks

This method returns the level of TAPI that is installed on the PC.

Return Value

Version of TAPI found on this PC.

CServiceProvider::GetUIManager

LPCTSTR GetUIManager() const;

Remarks

This method returns the string name of the user interface DLL that should be used for all UI related events. This is set by the constructor of the **CServiceProvider** object.

Return Value

Pointer to a NULL terminated string containing the name of the user interface DLL.

CServiceProvider::MatchTones

virtual bool MatchTones (DWORD *dwSFreq1***, DWORD** *dwSFreq2***,**
DWORD *dwSFreq3***, DWORD** *dwTFreq1***,**
DWORD *dwTFreq2***, DWORD** *dwTFreq3***);**

dwSFreq1	Search Frequency tone number 1
dwSFreq2	Search Frequency tone number 2
dwSFreq3	Search Frequency tone number 3
dwTFreq1	Target Frequency tone number 1
dwTFreq2	Target Frequency tone number 2
dwTFreq3	Target Frequency tone number 3

Remarks

This method compares a set of tones against each other. The search frequency is what the provider is looking for, the target being the frequency that the hardware detected. The default implementation simply compares the three together. This most likely will not be adequate for most providers and therefore this method should be overridden to support full tone frequency matching.

This method is called when tone monitoring is turned on for specific tones through **lineMonitorTones**.

Return Value

TRUE if the frequencies match, or FALSE if they do not.

CServiceProvider::ProcessCallParameters

virtual LONG ProcessCallParameters(CTSPILineConnection* *pLine***,**
LPLINECALLPARAMS *lpCallParams***);**

pLine	Line object that owns the call.
lpCallParams	**LINECALLPARAMS** structure from TAPI.

Remarks

This method is called whenever a **LINECALLPARAMS** structure is passed through the provider. It validates the structure against the line to ensure that the values for media, data rate, and dialing information are all within allowed values.

It may be overridden by derived providers to provide specialized handling of the **LINECALLPARAMS** structure.

Return Value

TAPI result code or FALSE if the structure was valid for the provider.

CServiceProvider::ReadProfileDWord

DWORD ReadProfileDWord (DWORD *dwPPid***, LPCTSTR** *pszEntry***,**
DWORD *dwDefault* **= 0);**

dwPPid	Provider ID assigned by TAPI to the device storing data.
pszEntry	Key to read from.
dwDefault	Default number to return if not found.

Remarks

This method reads a numeric value from the storage section devoted to the service provider in the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

Any data stored using this API will be automatically removed when the provider is de-installed using **TSPI_providerRemove**.

Return Value

DWORD read from persistent storage or default value if not found.

CServiceProvider::ReadProfileString

TString ReadProfileString (DWORD *dwPPid***, LPCTSTR** *pszEntry***,**
LPCTSTR *pszDefault* **= “”);**

dwPPid	Provider ID assigned by TAPI to the device storing data.
pszEntry	Key to read from.
pszDefault	Default string to return if not found.

Remarks

This method reads a string from the storage section devoted to the service provider in the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

Any data stored using this API will be automatically removed when the provider is de-installed using **TSPI_providerRemove**.

Return Value

String read from persistent storage or default value if not found.

CServiceProvider::RemoveTimedCall

void RemoveTimedCall(CTSPICallAppearance* *pCall***);**

pCall	The call object to remove from the internal array.

Remarks

This method is used to remove a call object from the internal array of calls that *need* timer events. This will cause the call's **OnTimer** method to be stop being called.

This is used internally by TSP++ to manage digit, tone, and media gathering events that have timeout conditions associated with them. It may be used by the derived provider in order to do some other form of work on the call by overriding the **CTSPICallAppearance::OnTimer** method.

CServiceProvider::RenameProfile

bool ReadProfileString (DWORD *dwPPid***, DWORD** *dwNewPPid***);**

dwPPid	Provider ID assigned by TAPI to the device storing data.
dwPPid	New Provider ID to move previous information into.

Remarks

This method copies all existing profile information stored under the listed provider id into a new section in the registry based on the new id. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

This method should be used if the provider supports dynamic creation of devices and therefore needs to move profile information around.

Any data stored using this API will be automatically removed when the provider is de-installed using **TSPI_providerRemove**.

Return Value

TRUE if the profile was renamed successfully. FALSE if the rename failed.

CServiceProvider::SetRuntimeObjects

Protected

void SetRuntimeObjects(LPCSTR *pszDevObj***,**
LPCSTR *pszLineObj* **= NULL, LPCSTR** *pszAddrObj* **= NULL,**
LPCSTR *pszCallObj* **= NULL, LPCSTR** *pszConfCall* **= NULL,**
LPCSTR pszPhoneObj **= NULL);**

pszDevObject	This is the class object that should be used in place of the standard device object. It must be derived from the **CTSPIDevice** object.
pszLineObject	This is the class object that should be used in place of the standard line device object. It must be derived from the **CTSPILineConnection** object.

pszAddrObject	This is the class object that should be used in place of the standard address object. It must be derived from the **CTSPIAddressInfo** object.
pszCallObject	This is the class object that should be used in place of the standard call appearance object. It must be derived from the **CTSPICallAppearance** object.
pszConfObject	This is the class object that should be used in place of the standard conference call object. It must be derived from the **CTSPIConferenceCall** object.
pszPhoneObject	This is the class object that should be used in place of the standard phone device object. It must be derived from the **CTSPIPhoneConnection** object.

Remarks

This method allows the derived provider to replace or supplement the existing functionality within each object type with derived objects. This method should be called within the constructor of the **CServiceProvider** object to override each desired class. If the method passed NULL in for any parameter, the default object type is used.

The given class name must be in ASCII (not Unicode) and must have a factory defined for it.

Note that this method should only be called from within the constructor of the service provider object. If you attempt to call the method at any other time, unpredictable results may occur.

CServiceProvider::TraceOut

virtual void TraceOut(TString& *strBuff***);**

strBuff	The string that was generated by the trace facility.

Remarks

This method is called to output an event into the persistent trace buffer. The actual string may be of any length.

This method is only called when the debug thread has been turned on by calling the global **ActivateTraceLog** function. It is always called in the context of a low-priority logger thread created by the TSP++ library.

The actual event will have been sent to the debug trace window using **OutputDebugString** before this is called. This method is simply to store the information in some type of persistent file or window.

CServiceProvider::WriteProfileDWord

bool WriteProfileDWord (DWORD *dwPPid***, LPCTSTR** *pszEntry***,**
DWORD *dwValue***);**

dwPPid	Provider ID assigned by TAPI to the device storing data.
pszEntry	Key to write to.
dwValue	Numeric value to write to the key.

Remarks

This method writes a numeric value into the storage section devoted to the service provider inside the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

Any data stored using this API will be automatically removed when the provider is de-installed using **TSPI_providerRemove**.

Return Value

TRUE if the value was stored successfully.

CServiceProvider::WriteProfileString

bool WriteProfileString (DWORD *dwPPid***, LPCTSTR** *pszEntry***,**
LPCTSTR *pszValue***);**

dwPPid	Provider ID assigned by TAPI to the device storing data.
pszEntry	Key to write to.
pszValue	String value to write to the entry.

Remarks

This method writes a string into the storage section devoted to the service provider inside the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

Any data stored using this API will be automatically removed when the provider is de-installed using **TSPI_providerRemove**.

Return Value

TRUE if the value was stored successfully.

CTSPIBaseObject

The **CTSPIBaseObject** is a basic synchronizable class defined in the TSP++ library. Each of the telephony objects (devices, lines, phones, requests, calls) are derived from this object in order to inherit basic properties common to all of them.

Object synchronization

Third party service providers, in general, should always be multi-threaded in order to maximize performance on the Windows NT platform.

The main problem introduced with multiple threads accessing the same objects is synchronization. Because many of the telephony objects have data structures that involve pointers and list traversal, some mechanism must be introduced into the model so that pointers remain valid during any access/write operations once it has been started. In the same fashion, arrays, once a traversal is initiated, cannot be changed without potentially causing a stray pointer access.

To fulfill this requirement in TSP++, each object derived from **CTSPIBaseObject** has a built in synchronization object that is used anytime a volatile operation is about to be performed on the object.

CEnterCode

The library defines an object called **CEnterCode** that takes as it's single parameter any **CTSPIBaseObject** (or derived object) and locks the object for update or volatile operation. This object may also be used in derived providers if they need to access internal data structures of one of their objects. See the reference section on *CEnterCode* for more information on this object.

Item Data

Each object derived from **CTSPIBaseObject** has a 32-bit item data value that may be used by the service provider for any purpose. To access it, use the **GetItemData** and **SetItemData** members.

Constructor and Destructor

CTSPIBaseObject	Constructor for the base object.
~CTSPIBaseObject	Destructor for the base object

Operations - Public Members

GetItemData	Returns the 32-bit item data value associated with this object as a DWORD value.
GetItemDataPtr	Returns the 32-bit item data value associated with this object cast to a void* pointer.
SetItemData	Sets the 32-bit item data value to a DWORD value
SetItemDataPtr	Sets the 32-bit item data value to a pointer.

Overridables - Public Members

Dump	Called to dump the object to a string.

CTSPIBaseObject::CTSPIBaseObject

CTSPIBaseObject()

Remarks

Constructor for the base object

CTSPIBaseObject::~CTSPIBaseObject

virtual ~CTSPIBaseObject()

Remarks

Destructor for the basic object. This must be overridden by any derived class to free data specific to the object.

CTSPIBaseObject::Dump

virtual TString Dump() const {

Remarks

This debug method returns a debug string with information about this object. It should be overridden by any derived object to return valid information based on the type of object.

Return Value

A string object. Note this method returns the value "This object has no dump output".

CTSPIBaseObject::GetItemData

DWORD GetItemData() const;

Remarks

This method returns the 32-bit item data associated with the object as a numeric non-pointer value. If the value is a pointer, use the **GetItemDataPtr** retrieval method.

Return Value

32-bit item data value

CTSPIBaseObject::GetItemDataPtr

void* GetItemDataPtr() const;

Remarks

This method returns the 32-bit item data associated with the object as a pointer value. If the value is not a pointer, use the **GetItemDataPtr** retrieval method.

Return Value

32-bit item data value

CTSPIBaseObject::SetItemData

void SetItemData(DWORD *dwItem***);**

dwItem	Value to store into the object's item data.

Remarks

This method sets the 32-bit item data associated with the object to a numeric non-pointer value. If the value is a pointer, use the **SetItemDataPtr** method.

CTSPIBaseObject::SetItemDataPtr

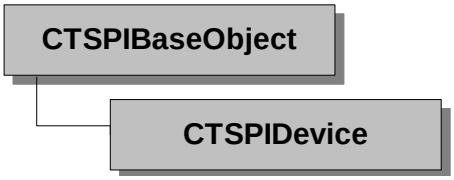
void SetItemDataPtr(void* *pvItem***);**

pvItem	Value to store into the object's item data.

Remarks

This method sets the 32-bit item data associated with the object to a 32-bit pointer value. If the value is not a pointer, use the **SetItemData** method.

CTSPIDevice



The **CTSPIDevice** object maintains the information specific to a particular TAPI device. Each device installed into TAPI will be assigned a unique 32-bit identifier called a *Provider ID*. This identifies the device *and* provider to TAPI. Each provider device may then manage connections to the physical telephony network. The device is represented in the TSP++ library as a **CTSPIDevice** object. In general there will only be one **CTSPIDevice** in a service provider, but this is not a rule, multiple devices can be supported through a single provider in TAPI.

Device Initialization

When the device object is first constructed by the **CServiceProvider::providerInit** method, it creates line and phone device objects to control the lines and phones that are present on the device. The number of objects created is passed to the device constructor and is supplied by TAPI by the **CServiceProvider::providerEnumDevices** method (which normally reads it from the registry)

Lines and Phones

The device object is responsible for maintaining the list of line and phone devices that are present on this physical connection to the network. The device object creates each line and phone object during the initialization process. The number of lines and phones present is determined either by the service provider or by TAPI during initialization and is generally static during the life of the provider.

If lines and phones are dynamic, then the provider must support the Plug & Play features to notify TAPI about line and phone additions. This support is provided through the **CreateLine** and **CreatePhone** methods of the device object. Removal of a line or phone is performed through the **RemoveLine** and **RemovePhone** methods.

Device Open/Close events

The device object may be used to control access to the physical telephony device if desired. When any line or phone connection is opened by TAPI, the **OpenDevice** method will be invoked on the device. This may occur more than once while the TSP is open, so if only one physical connection is established, reference counting must be performed. When a line or phone is closed, the **CloseDevice** method is called. By overriding these two methods, all open/close events for all lines and phones for a single device may be handled in one place.

If a full PBX TSP implementation is being designed, then it is better to open the device during the **CTSPIDevice::Init** processing so that events from the switch will always be seen even if no line is open on the provider. This will ensure that events are not lost due to a lack of connection.

Thread Pool Management

TSP++ Version 3.0 introduces a new worker thread management template that can be used by derived service providers. It is meant to be used within the **CTSPIDevice** derived object (although that is not required).

Locking Keys

The template class generates a C++ object which supports a *key* entry that is used to synchronize requests with each other – as long as the key is *locked*, any events which share that key will not be processed by worker threads. Once the key is unlocked, then next pending event for that key (if any exists in the queue) will be processed. If another event is queued which is associated with a different key element that is not locked, it will be processed immediately – while the other request is running. While the requests are waiting they have no thread paused on them (i.e. the library does not pause threads on queued requests – worker threads scan the queue to find an appropriate request and pull it out when one is found).

Using the template

To use the template, include the *poolmgr.h* header file and instantiate an object within the derived device object framework. The template arguments are:

```
template <class _KEY, class _REQ >
class CThreadPoolMgr : public CIntThreadMgr
```

where **_KEY** is the class or type to use as the locking key (see above)
_REQ is the class or type representing an event from the switch

The constructor of the device object should then initialize the pool manager by passing a maximum count of threads to create. The default value of zero will cause the pool manager to create 1 thread per megabyte of RAM on the server with a maximum of 32 threads per processor on the server.
When the device receives an event, the thread should call the **Add** method of the pool manager object to queue the event for a worker thread. This method is prototyped as:

```
bool Add(CTSPIConnection* pConn, _KEY key, _REQ request);
```

where **pConn** is the line or phone connection to queue the event for
key is the locking key to check before allowing the event to run
request is the actual event object, type or structure.

For more information on this subject, see the reference section on the *Thread Pool Manager*.

Constructor and Destructor

CTSPIDevice	Constructs a **CTSPIDevice** object.
~CTSPIDevice	Destructor that deallocates all the information in the device object.

Operations - Public Methods

AddAgentActivity	Adds a new agent activity to the device.
AddAgentGroup	Adds a new agent group to the device.
AddAgentSpecificExtension	Adds a new agent extension to the device.
AllocStream	Returns a **TStream** to restore persistent object information from.
AssociateLineWithPhone	This connects a line to a phone device.
CreateLine	Dynamically add a new line device to TAPI
CreatePhone	Dynamically add a new phone device to TAPI.
DoesAgentActivityExist	Returns a true/false result if the passed activity id is valid.
DoesAgentGroupExist	Returns a true/false result if the passed group id is valid.
DoesAgentSpecificExtensionExist	Returns a true/false result if the agent extension is valid.
FindCallHub	Returns the call hub object for the given call-id
FindLineConnectionByDeviceID	Return the line object based on TAPI line device id.
FindPhoneConnectionByDeviceID	Return the phone object based on TAPI phone device id.
FindLineConnectionByPermanentID	Returns the line object based on the permanent device id.
FindPhoneConnectionByPermanentID	Returns the phone object based on the permanent device id.
GetAgentActivity	Returns an agent activity based on the sequential index.
GetAgentActivityById	Returns an agent activity string from the unique identifier.
GetAgentActivityCount	Returns the number of agent activities defined in the system.
GetAgentGroup	Returns an agent group based on the sequential index.
GetAgentGroupById	Returns an agent group name based on the unique identifer.
GetAgentGroupCount	Returns the number of agent groups defined

	in the system
GetAgentSpecificExtension	Returns an agent-specific extension based on the index.
GetAgentSpecificExtensionCount	Returns the number of defined agent-specific extensions.
GetLineCount	Return the total number of line devices.
GetLineConnectionInfo	Return the line object associated with an index.
GetPhoneCount	Return the total number of phone devices.
GetPhoneConnectionInfo	Return the phone object associated with an index.
GetProviderID	Return the TAPI assigned provider id for this device.
GetProviderHandle	Returns the provider handle assigned by TAPI.
GetSwitchInfo	Returns the string used to describe this device.
OnAsynchRequestComplete	This function allows the TSP to explicitely complete a request by requested vs. using a request object.
RemoveAgentActivity	Removes an agent activity from the device.
RemoveAgentGroup	Removes an agent group from the device.
RemoveAgentSpecificExtension	Removes an agent extension from the device.
RemoveLine	Remove an existing line from the device.
RemovePhone	Remove an existing phone from the device.
SetIntervalTimer	Turns the interval timer for this device on or off.
SetSwitchInfo	Sets the textual information used to describe this device

Operations - Protected Methods

RunTimer	This function processes the internal interval timer used to perform periodic tasks with in the TSP library.

Overridables - Public Members

CloseDevice	Called by the **CTSPIConnection::CloseDevice** method.
GenericDialogData	Processes dialog user-interface events.
OpenDevice	Called by the **CTSPIConnection::OpenDevice** method. Default implementation passes control to **CServiceProvider::OpenDevice**.

read	Reads configuration information about the device from the registry using a registry iostream. This is only called if the SPLUI library is used to store object information in the registry.
ReceiveData	This routes an event to a line or phone device based on a permanent line or phone device identifier or to all lines/phones.

Overridables - Protected Members

Init	Initialization method called after the constructor.
OnAsynchRequestComplete	Called when any asynchronous request completes. Default implementation notifies TAPI through the asynchronous callback.
OnCancelRequest	Called by the **CTSPIConnection** object when a request is being canceled.
OnNewRequest	Called by the **CTSPIConnection** object when a new request is created and being added to the device list.
OnTimer	Called by the **CServiceProvider DistributeIntervalTimer** method. Default implementation gives a periodic timer to all connections on the device.

CTSPIDevice::AddAgentActivity

int AddAgentActivity(DWORD *dwID***, LPCTSTR** *pszName***);**

dwID	Unique agent activity identifier.
pszName	Text name of the activity.

Remarks

This method adds a new agent activity to the device. This activity will then be available to any line device that supports agents. It will be reported in the agent capabilities as a valid activity.

Return Value

Numeric index for the agent activity. (-1) if there was an error.

CTSPIDevice::AddAgentGroup

int AddAgentGroup(LPCTSTR *pszName***, DWORD** *dwGroupID1***,**
DWORD *dwGroupID2***=0, DWORD** *dwGroupID3***=0,**
DWORD *dwGroupID4***=0);**

pszName	Text name of the new agent group.
dwGroupID1	First 32-bit value of the group identifier.
dwGroupID2	Second 32-bit value of the group identifier.
dwGroupID3	Third 32-bit value of the group identifier.
dwGroupID4	Fourth 32-bit value of the group identifier.

Remarks

This method adds a new agent group to the device. This group will then be available to any line device that supports agents. It will be reported in the agent capabilities as a valid group.

Return Value

Numeric index for the agent group. (-1) if there was an error.

CTSPIDevice::AddAgentSpecificExtension

int AddAgentSpecificExtension(DWORD *dwID1***, DWORD** *dwID2***=0,**
DWORD *dwID3***=0, DWORD** *dwID4***=0);**

dwID1	First 32-bit value of the extension identifier.
dwID2	Second 32-bit value of the extension identifier.
dwID3	Third 32-bit value of the extension identifier.
dwID4	Fourth 32-bit value of the extension identifier.

Remarks

This method adds a new agent specific extension identifier to the device. This extension will then be available to any line device that supports agents. It will be reported in the agent capabilities as a valid agent extension.

Agent specific extensions are used to extend the agent capabilities supported by TAPI to include features not directly addressed in the TAPI specification.

Return Value

Numeric index for the agent extension. (-1) if there was an error.

CTSPIDevice::AllocStream

TStream* AllocStream();

Remarks

This method is called when the provider is loading or saving its persistent object information. The default implementation returns a stream of type **TRegstream**. It may be overridden to provide a different stream implementation. This function is called in the context of the **TSPI_providerInit** function if there are line or phone counts in the registry. For information on using this function, see the *User's Guide* and the section on *Persistent Object Information*.

Return Value

Stream object allocated on the heap to load information for the provider.

CTSPIDevice::AssociateLineWithPhone

void AssociateLineWithPhone(unsigned int *iLine***, unsigned int** *iPhone***);**

iLine	Index of a CTSPILineConnection object.
iPhone	Index of a CTSPIPhoneConnection object.

Remarks

This method associates a line and a phone object together so that TAPI applications can identify a phone for a particular station and vice-versa.

This method creates the device class "tapi/line" on the phone device and the device class "tapi/phone" on the line device per the TAPI specification.

CTSPIDevice::CloseDevice

virtual bool CloseDevice (CTSPIConnection* *pConn***);**

pConn	Line or phone connection that is requesting close.

Remarks

This method is called from the **CTSPILineConnection** and **CTSPIPhoneConnection** objects to close the physical device.

The default implementation returns FALSE.

Return Value

TRUE/FALSE success indicator.

CTSPIDevice::CreateLine

CTSPILineConnection* CreateLine(DWORD *dwItemData***);**

dwItemData Value to pass to the CTSPILineConnection::Init method.

Remarks

This method dynamically adds a line to the device array. It is only available if the TSP exports the **TSPI_providerEnumDevices** method to allow the class library to get a pointer to the line creation callback.

As soon as this method returns, the line is available from within the service provider. TAPI will notify all running applications that a new line has been created, and eventually will assign the new line a permanent line identifier. TAPI will notify the provider about the new line identifier through a callback that indicates that the line is fully ready for use by applications. This notification is managed within the library automatically.

Return Value

The return value is the pointer of the newly created line device or NULL if the creation failed.

CTSPIDevice::CreatePhone

CTSPIPhoneConnection* CreatePhone(DWORD *dwItemData***);**

dwItemData Value to pass to the CTSPIPhoneConnection::Init method.

Remarks

This method dynamically adds a telephone to the device array. It is only available if the TSP exports the **TSPI_providerEnumDevices** method to allow the class library to get a pointer to the phone creation callback.

As soon as this method returns, the phone is available from within the service provider. TAPI will notify all running applications that a new phone has been created, and eventually will assign the new phone a permanent phone identifier. TAPI will notify the provider about the new phone identifier through a callback that indicates that the phone is fully ready for use by applications. This notification is managed within the library automatically.

Return Value

The return value is the pointer to the newly created phone device or NULL if it could not be created.

CTSPIDevice::DoesAgentActivityExist

bool DoesAgentActivityExist(DWORD *dwActivity***) const;**

dwActivity Agent activity identifer

Remarks

This method checks the list of valid agent activities and returns whether the given activity exists on this device.

Return Value

TRUE if the activity exists, FALSE if it does not.

CTSPIDevice::DoesAgentGroupExist

bool DoesAgentGroupExist(DWORD *dwGroupID1***, DWORD** *dwGroupID2***=0, DWORD** *dwGroupID3***=0, DWORD** *dwGroupID4***=0) const;**

dwGroupID1	First 32-bit value of the group identifier.
dwGroupID2	Second 32-bit value of the group identifier.
dwGroupID3	Third 32-bit value of the group identifier.
dwGroupID4	Fourth 32-bit value of the group identifier.

Remarks

This method checks the list of valid agent groups and returns whether the given group exists on this device.

Return Value

TRUE if the group exists, FALSE if it does not.

CTSPIDevice::DoesAgentSpecificExtensionExist

bool DoesAgentSpecificExtensionExist(DWORD *dwID1***, DWORD** *dwID2***=0, DWORD** *dwID3***=0, DWORD** *dwID4***=0) const;**

dwID1	First 32-bit value of the extension identifier.
dwID2	Second 32-bit value of the extension identifier.
dwID3	Third 32-bit value of the extension identifier.
dwID4	Fourth 32-bit value of the extension identifier.

Remarks

This method checks the list of valid agent specific extensions and returns whether the given extension exists on this device.

Return Value

TRUE if the extension exists, FALSE if it does not.

CTSPIDevice::FindCallHub

const CTSPICallHub* FindCallHub(DWORD *dwCallID***) const;**

dwCallID	Call identifier to locate a call hub object for.

Remarks

This method returns the **CTSPICallHub** object for the given call identifier. If the call hub object does not exist (i.e. no calls are in existence with the given call-id) then NULL is returned.

This allows the provider to quickly find a list of calls matching a given call identifier. For information on the call hub object, see the reference section on *CTSPICallHub*.

Return Value

A pointer to the call hub object or NULL if there are no calls matching the given call-id.

CTSPIDevice::FindLineConnectionByDeviceID

CTSPILineConnection* FindLineConnectionByDeviceID(
DWORD *dwDeviceID*) **const;**

dwDeviceID TAPI-assigned device identifier for line device.

Remarks

This method runs through the line device array and searches for the **CTSPILineConnection** that is matched to the specified TAPI device id.

Each line and phone device is assigned a unique index within the TAPI system, based on the order in which it was added to TAPI. The number is not guaranteed to be sequential within the provider (although generally will be).

Return Value

The line connection object that was found to match the device id or NULL if it could not be found.

CTSPIDevice::FindLineConnectionByPermanentID

CTSPILineConnection* FindLineConnectionByPermanentID(
DWORD *dwConnID*) **const;**

dwConnID Permanent line identifier to find a line for.

Remarks

This method runs through the line device array and searches for the **CTSPILineConnection** that matches to the specified permanent line identifier.

Each line and phone device can be assigned a permanent numeric identifier. The default value assigned by TSP++ is a combination of the provider id and the index within the line device array. Normally this would be changed by a derived provider to reflect some associative value for the actual switch station device (such as a station identifier, queue number, etc.)

Return Value

The line connection object that was found to match the permanent line id or NULL if it could not be found.

CTSPIDevice::FindPhoneConnectionByDeviceID

CTSPIPhoneConnection* FindPhoneConnectionByDeviceID(
DWORD *dwDeviceID*) **const;**

dwDeviceID TAPI-assigned device identifier for phone device.

Remarks

This method runs through the phone device array and searches for the **CTSPIPhoneConnection** that is matched to the specified TAPI device id.

Each line and phone device is assigned a unique index within the TAPI system, based on the order in which it was added to TAPI. The number is not guaranteed to be sequential within the provider (although generally will be).

Return Value

The phone connection object that was found to match the device id or NULL if it could not be found.

CTSPIDevice::FindPhoneConnectionByPermanentID

CTSPIPhoneConnection* FindPhoneConnectionByPermanentID(
DWORD *dwConnID***) const;**

dwConnID Permanent phone identifier to find a line for.

Remarks

This method runs through the phone device array and searches for the **CTSPIPhoneConnection** that matches to the specified permanent phone identifier.

Each line and phone device can be assigned a permanent numeric identifier. The default value assigned by TSP++ is a combination of the provider id and the index within the phone device array. Normally a derived provider would change this value to reflect some associative value for the actual switch station device (such as a station identifier).

Return Value

The phone connection object that was found to match the permanent phone id or NULL if it could not be found.

CTSPIDevice::GenericDialogData

virtual LONG GenericDialogData (LPVOID *lpParam***, DWORD** *dwSize***);**

lpParam Parameter from the UI dialog.
dwSize Size of the passed parameter block.

Remarks

This method is called when the user-interface component of the TSP is sending data back to the service provider, and the object type specified in the **TSPI_providerGenericDialogData** function was set to **TUISPIDLL_OBJECT_PROVIDERID**.

Return Value

TAPI result code or FALSE for success.

CTSPIDevice::GetAgentActivity

const TAgentActivity* GetAgentActivity(unsigned int *iPos*) const;

iPos	Numeric index position of the agent activity.

Remarks

This method returns an agent activity structure from the numeric index array position. This should be between zero and **GetAgentActivityCount**.

Return Value

A pointer to the agent activity structure at the given array position, NULL if no activity exists at that position.

CTSPIDevice::GetAgentActivityByID

TString GetAgentActivityById(DWORD *dwID*) const;

dwID	Unique agent activity identifier

Remarks

This method returns the agent activity structure that is associated with the given agent activity identifier. The activity should have been added using the **AddAgentActivity** method.

Return Value

A pointer to the agent activity structure associated with the given identifier or NULL if no activity is associated with that identifier.

CTSPIDevice::GetAgentActivityCount

unsigned int GetAgentActivityCount() const;

Remarks

This method returns the number of agent activities that are associated with this device.

Return Value

The numbers of agent activities on this device or zero if no activities are defined on this device.

CTSPIDevice::GetAgentGroup

const TAgentGroup* GetAgentGroup(unsigned int *iPos*) const;

iPos	Numeric index position of the agent group.

Remarks

This method returns an agent group structure from the numeric index array position. This should be between zero and **GetAgentGroupCount**.

Return Value

A pointer to the agent group structure at the given array position, NULL if no group exists at that position.

CTSPIDevice::GetAgentGroupByID

TString GetAgentGroupById(DWORD *dwGroupID1***, DWORD** *dwGroupID2***=0, DWORD** *dwGroupID3***=0, DWORD** *dwGroupID4***=0) const;**

dwGroupID1	First 32-bit value of the group identifier.
dwGroupID2	Second 32-bit value of the group identifier.
dwGroupID3	Third 32-bit value of the group identifier.
dwGroupID4	Fourth 32-bit value of the group identifier.

Remarks

This method returns the agent group structure that is associated with the given agent group identifier. The group should have been added using the **AddAgentGroup** method.

Return Value

A pointer to the agent group structure associated with the given identifier or NULL if no group is associated with that identifier.

CTSPIDevice::GetAgentGroupCount

unsigned int GetAgentGroupCount() const;

Remarks

This method returns the number of agent groups that are associated with this device.

Return Value

The numbers of agent groups on this device or zero if no groups are defined on this device.

CTSPIDevice::GetAgentSpecificExtension

const TAgentSpecificEntry* GetAgentSpecificExtension(
unsigned int *iPos***) const;**

iPos	Numeric index position of the agent extension.

Remarks

This method returns an agent extension structure from the numeric index array position. This should be between zero and **GetAgentSpecificExtensionCount**.

Return Value

A pointer to the agent extension structure at the given array position, NULL if no extension exists at that position.

CTSPIDevice::GetAgentSpecificExtensionCount

unsigned int GetAgentSpecificExtensionCount() const;

Remarks

This method returns the count of agent extension structures defined on this device.

Return Value

The numeric count of agent specific extensions added to the device using **AddAgentSpecificExtension** or zero if no extensions are defined.

CTSPIDevice::GetLineConnectionInfo

CTSPILineConnection* GetLineConnectionInfo(int *nIndex*) const;

nIndex	Zero-based index of the line to retrieve. This should not exceed the value returned by **GetLineCount**.

Remarks

This method returns **CTSPILineConnection** object that is at the specified array position.

Return Value

Line object that is at the specified array position or NULL if no line is available at that position.

CTSPIDevice::GetLineCount

int GetLineCount() const;

Remarks

This method returns the number of lines that are in the internal provider line array. This is indicative of the number of lines initially assigned to the provider, along with any dynamically added lines while the provider has been running.

Return Value

Count of lines present in the provider. This will always be one more than the highest retrievable index using **GetLineConnectionInfo**.

Returns zero if no phones are available.

CTSPIDevice::GetPermanentDeviceID

DWORD GetPermanentDeviceID() const;

Remarks

This method returns a unique device identifier within this provider that is used to create unique line/phone indexes to pass through to the companion application.

This identifier is created using the *permanent provider identifier* that is assigned by TAPI to the device during **TSPI_providerInstall**. The permanent provider id is converted to a 16-bit value and placed in the high-order word of the returning device id. The low-order word is assigned by each of the line or phone devices to further identify an object owned by this device.

The return value from this method can be used to mask out lines and phone devices when multiple **CTSPIDevice** objects are supported within a single provider.

Return Value

Permanent provider identifier for this device.

CTSPIDevice::GetPhoneConnectionInfo

CTSPIPhoneConnection* GetPhoneConnectionInfo(int *nIndex***) const;**

nIndex	Zero-based index of the phone to retrieve. This should not exceed the value returned by **GetPhoneCount**.

Remarks

This method returns **CTSPIPhoneConnection** object that is at the specified array position.

Return Value

Phone object that is at the specified array position or NULL if no phone is available at that position.

CTSPIDevice::GetPhoneCount

int GetPhoneCount() const;

Remarks

This method returns the number of phones that are in the internal provider phone array. This is indicative of the number of phones initially assigned to the provider, along with any dynamically added phones while the provider has been running.

Return Value

Count of phones present in the provider. This will always be one more than the highest retrievable index using **GetPhoneConnectionInfo**.

Returns zero if no phones are available.

CTSPIDevice::GetProviderHandle

HPROVIDER GetProviderHandle() const;

Remarks

This method returns the provider handle assigned by TAPI to this provider/device.

Return Value

Provider handle assigned to the device during **TSPI_providerInit**.

CTSPIDevice::GetProviderID

DWORD GetProviderID() const;

Remarks

This method returns the unique device identifier that was assigned to this provider/device combination. This corresponds to the *permanent provider identifier* that is used in the **ReadProfileXX** and **WriteProfileXX** methods in the **CServiceProvider** object.

This identifier is assigned by TAPI to the device during **TSPI_providerInstall**, and will *always* be the same – until the provider is de-installed.

Since the TSP++ library supports multiple devices within a single provider shell, the provider id is somewhat of a misnomer. It really represents a combination of provider and device within the provider to the library.

Return Value

Permanent provider identifier for this device.

CTSPIDevice::GetSwitchInfo

LPCTSTR GetSwitchInfo() const;

Remarks

This method returns the textual switch information that is associated with this device. This information is set using the **SetSwitchInfo** method on the device object. The switch information is reported in the **LINEDEVCAPS** and **PHONEDEVCAPS** of all line and phone devices associated with the given device object.

There is no standard for this value – it should simply be information about the hardware switch or device the TSP is connected to. Normally this would be manufacturer information, ACD software revision, etc.

Return Value

Pointer to string containing the switch information.

CTSPIDevice::Init

Protected

virtual void Init(DWORD *dwProviderId*, **DWORD** *dwBaseLine*,
DWORD *dwBasePhone*, **DWORD** *dwLines*, **DWORD** *dwPhones*,
HPROVIDER *hProvider*, **ASYNC_COMPLETION** *lpfnCompletion*);

dwProviderId	Permanent provider identifier assigned by TAPI.
dwBaseLine	Base index of first line to create
dwBasePhone	Base index of first phone to create
dwLines	Number of lines to initialize.

dwPhones	Number of phones to initialize
hProvider	Provider handle assigned by TAPI
lpfnCompletion	Asynchronous completion callback pointer.

Remarks

This method is called directly after the constructor to initialize the provider/device. It is called during the **TSPI_providerInit** function by the **CServiceProvider** object. It is responsible for setting up the device and creating all the line and phone objects.

If this method is overridden, you *must* call the base class implementation.

CTSPIDevice::OnAsynchRequestComplete

void OnAsynchRequestComplete(DRV_REQUESTID *drvRequestID*,
LONG *lResult*);

drvRequest	TAPI assigned request identifier
lResult	Final return result of the completed request.

Remarks

This method allows the derived TSP to explicitely complete a request by requested rather than using the request object. Note that if a request object is associated with the request it will **not** be updated.

This is best used for lineDevSpecific completions.

CTSPIDevice::OnAsynchRequestComplete

Protected

virtual void OnAsynchRequestComplete(LONG *lResult* **= 0L,**
CTSPIRequest* *pRequest* **= NULL);**

lResult	Final return result of the completed request.
pRequest	Request object that has completed.

Remarks

This method is called by the **CTSPIRequest** object when any request on the device completes. It gives the device an opportunity to do any post-request cleanup or simply monitor the completions as they occur.

If you override this method, you *must* call the base class implementation.

CTSPIDevice::OnCancelRequest

Protected

virtual void OnCancelRequest (CTSPIRequest* *pRequest*);

pRequest	Request packet that is being cancelled.

Remarks

This method is called by the phone or line object when a request is canceled and it has already been started on the device. It gives the device object an opportunity to examine the request and do some post-processing before it is deleted.
The default behavior is to pass control to the **CServiceProvider::CancelRequest** method.

CTSPIDevice::OnNewRequest

Protected

virtual bool OnNewRequest (CTSPIConnection* *pConn*, **CTSPIRequest*** *pRequest*);

pConn	Connection that the request is starting on.
pRequest	Request object that is about to be added to the list.

Remarks

This method is called by the connection object each time a new request packet is added to the request list. It is called before the request is officially added to the list and allows the device object to manipulate the list before insertion.

The default behavior is to pass control to the **CServiceProvider::OnNewRequest** method.

Return Value

TRUE if the connection object should continue to add the request. FALSE if the request is to be canceled.

CTSPIDevice::OnTimer

Protected

virtual void OnTimer();

Remarks

This method is called from the interval timer thread that is turned on using the **SetIntervalTimer** method.

If the method is not overridden, it will run through all the lines and phones in the object array and pass the timer to each of them.

CTSPIDevice::OpenDevice

virtual bool OpenDevice (CTSPIConnection* *pConn*);

pConn	Line or phone connection that is requesting open.

Remarks

This method is called from the **CTSPILineConnection** and **CTSPIPhoneConnection** objects to open the physical device.

If the method is not overridden, it will forward the request onto **CServiceProvider::OpenDevice**.

Return Value

TRUE if the device was opened successfully.

FALSE if the device failed to open, the **TSPI_lineOpen** or **TSPI_phoneOpen** that generated the request will fail and return an error.

CTSPIDevice::read

virtual std::istream& read(std::istream& *istm***);**

istm	input iostream to read information from.

Remarks

This method is called during initialization if it is determined that the device object information is contained within the registry. This is only used if the SPLUI library stored object information in the registry.

It may be overridden to read additional information from the stream (other than the information placed there by TSP++).

Note: you *must* retrieve information in the same order as it was stored in the SPLUI user-interface DLL implementation for your provider! If the TSP locks up on loading then check the serialization to ensure you are not reading past the iostream.

CTSPIDevice::ReceiveData

virtual void ReceiveData (DWORD *dwConnID***, LPCVOID** *lpBuff***);**

dwConnID	Connection ID which data is being routed to. This is a combination of the return value from permanent device id and the line/phone index in the internal array.
lpBuff	Pointer to buffer with received data.

Remarks

This method may be called to direct a received response to the appropriate line or phone device. The device object will determine the appropriate line or phone device using the **dwConnID** parameter. The LOWORD of the number is the array index of the line or phone. The method will then invoke the **ReceiveData** method of the object it located.

If the connection id is zero, then the device will route the data request through each line in turn, and then through each phone until one of them responds with a **TRUE** result (indicating that it processed the result) or the device runs out of lines and phones.

This method is not used by the TSP++ library internally, it is provided for derived providers if they want to route requests through the device object. In most implementations, the device object will route events using a thread pool object and this method will not be used.

CTSPIDevice::RemoveAgentActivity

void RemoveAgentActivity(DWORD *dwID***);**

dwID	Agent activity to remove from the system.

Remarks

This method removes an existing agent activity from the device. TAPI will be informed that the agent capabilities have been changed

CTSPIDevice::RemoveAgentGroup

void RemoveAgentGroup(DWORD *dwGroupID1***, DWORD** *dwGroupID2***=0, DWORD** *dwGroupID3***=0, DWORD** *dwGroupID4***=0);**

dwGroupID1	First 32-bit value of the group identifier.
dwGroupID2	Second 32-bit value of the group identifier.
dwGroupID3	Third 32-bit value of the group identifier.
dwGroupID4	Fourth 32-bit value of the group identifier.

Remarks

This method removes an existing agent group from the device. TAPI will be informed that the agent capabilities have been changed

CTSPIDevice::RemoveAgentSpecificExtension

void RemoveAgentSpecificExtension(DWORD *dwID1***, DWORD** *dwID2***=0, DWORD** *dwID3***=0, DWORD** *dwID4***=0);**

dwID1	First 32-bit value of the extension identifier.
dwID2	Second 32-bit value of the extension identifier.
dwID3	Third 32-bit value of the extension identifier.
dwID4	Fourth 32-bit value of the extension identifier.

Remarks

This method removes an existing agent specific extension from the device. TAPI will be informed that the agent capabilities have been changed

CTSPIDevice::RemoveLine

void RemoveLine(CTSPILineConnection* *pLine***);**

pLine	Line device object to remove from the system.

Remarks

This method dynamically removes a line from the device array. It is only available if the TSP is running under TAPI 2.0 or better and the provider exports the **TSPI_providerEnumDevices** method to allow the class library to get a pointer to the line creation/removal callback.

The line is not de-allocated or physically removed from the line array until the provider shuts down. This is to keep the same array indexes throughout the life of the library. You should no longer process requests for the line or access this object once this completes.

CTSPIDevice::RemovePhone

void RemovePhone(CTSPIPhoneConnection* *pPhone***);**

pPhone Phone device object to remove from the system.

Remarks

This method dynamically removes a phone from the device array. It is only available if the TSP is running under TAPI 2.0 or better and the provider exports the **TSPI_providerEnumDevices** method to allow the class library to get a pointer to the phone creation/removal callback.

The phone is not de-allocated or physically removed from the line array until the provider shuts down. This is to keep the same array indexes throughout the life of the library. You should no longer process requests for the phone or access this object once this completes.

CTSPIDevice::RunTimer

Protected

void RunTimer();

Remarks

This method implements the interval timer thread that is turned on using the **SetIntervalTimer** method. It calls the overridable **OnTimer** method and performs some required periodic housekeeping for TSP++.

CTSPIDevice::SetIntervalTimer

void SetIntervalTimer(DWORD *dwTimeout***=0);**

dwTimeout Timeout in milliseconds for the device interval timer.

Remarks

This method turns the interval timer on and off for the device. By default the timer is turned off. If this method is called with a non-zero value, the TSP++ library will create an interval timer thread and will invoke each of the line and phone objects **OnTimer** method on the specified interval.

If zero is specified, the thread is destroyed and the timer is turned off.

The thread is automatically turned off when the TSP is unloaded by TAPI.

CTSPIDevice::SetSwitchInfo

void SetSwitchInfo(LPCTSTR *pszSwitchInfo***);**

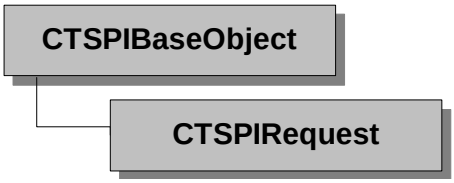
pszSwitchInfo Textual switch information to associate with this device

Remarks

This method sets the textual switch information that is associated with this device. The switch information is reported in the **LINEDEVCAPS** and **PHONEDEVCAPS** of all line and phone devices associated with the given device object.

There is no standard for this value – it should simply be information about the hardware switch or device the TSP is connected to. Normally this would be manufacturer information, ACD software revision, etc.

CTSPIRequest



The **CTSPIRequest** object maintains the status of a pending asynchronous TAPI operation.

Asynchronous Requests

The interactive nature of telephony requires that the TSP operate as if it were in a real-time operating environment. Many of the service provider methods are required to complete quickly and return their results to the TAPI server synchronously. Other functions (such as dialing or answering) may not be able to complete as quickly and therefore operate asynchronously.

When an operation completes asynchronously it performs part of its processing in the function call made by the application and the remainder of it in an independent execution thread after the application has returned from the function call. First, a request packet is created from the request and placed into the line or phone request queue. A **DRV_REQUESTID** is returned to the calling thread (TAPI) to indicate that the operation is being processed. Eventually, a worker thread is assigned to the request and interacts with the hardware to fulfill it.

Once the asynchronous request completes, the service provider calls the TAPI server through a callback function and reports that *"Event X has completed"*. Passed to this callback is the **DRV_REQUESTID** and a final result code for the operation indicating whether it was successful or not.

Request Management

Each asynchronous request initiated by the TAPI server will cause a request object to be created by the line or phone device that is being targeted, and inserted into the asynchronous request list of the connection object. If there are no pending requests active, then the newly inserted request will be started by calling the **ReceiveData** method of the line or phone object which is associated with the request. Otherwise, the request will simply be queued until the provider has completed all previously pending requests on the device. Since the asynchronous list is maintained at the connection level (line and phone), there generally will not be more than one or two pending request at a time. Requests to drop a call are considered special requests by the connection object and are always inserted at the front of the queue and started immediately

As each request completes, the service provider should call the **CTSPIConnection**'s **CompleteRequest** or **CompleteCurrentRequest** methods to inform TAPI that the request is finished and to delete the request from the list. If a request is canceled, or the

provider wishes to remove all pending requests based on some criteria, the **RemoveRequest** and **RemovePendingRequests** methods may be used. Specific requests may be located through the **FindRequest** method.

Processing a request

Some requests may require multiple operations to be performed with the telephone network to complete. For instance, when a dial request is received for a modem device, it must first take the modem off-hook, wait for a response, and then send a dial string. To handle this multiple-step processing, each request object has a *request-state* associated with it. Initially, the state starts out as **STATE_INITIAL**, and from that point forward, it is up to the service provider to define and change the state of each request.

For example, in a simple modem driver such as Microsoft's Unimodem, on **STATE_INITIAL**, it would take the modem device off-hook. The state in the associated **CTSPIRequest** object would then be changed using the **SetState** method, to be state number 2 (defined by Unimodem). Control would then exit, and when the modem sends the response, the request packet would again be processed, and the service provider would determine that we left off at state number 2. At this point, the result would be checked for an error, and if it was OK, the service provider would send a dial string and set the state to state number 3. Each request would be handled in this fashion. Once the service provider determines that the modem really did dial, it would complete the request. Then the request would be deleted and removed from the queue. The service provider would get the next request off the list and begin to process it in the same fashion.

Handling requests

Requests are routed directly to the line or phone objects that *own* them. In each case, the device object should call the **ReceiveData** method of the line or phone object so that the request may be processed.

Automatic Request Routing

TSP++ Version 3.0 introduces a new request/event routing mechanism similar to the Microsoft Foundation Class message map architecture.

In this routing mechanism, the line and phone class defines a *routing table* which associates a numeric request identifier (defined by TSP++ for each valid TSPI request) and a worker function to process that particular TSPI request.

The routing table is created in the derived service provider code using a series of macros:

DECLARE_TSPI_REQUESTS	This is used in the line or phone class definition and is used to create the internal variables which manage the routing table.
BEGIN_TSPI_REQUEST	This starts a routing table definition for lines or phones.
ON_TSPI_REQUEST	This joins a TAPI request with a member function of the line or phone class. There is a generic macro and specialized macros for each request type.

ON_TSPI_REQUEST_AUTO	This causes the given TAPI request to be completed automatically with no processing by the derived provider.
END_TSPI_REQUEST	This ends the routing table definition.

Usage of the routing macros

The routing map is global to the line or phone class it is defined for so there can only have one map for lines and one for phones.
To use these macros, place a **DECLARE_TSPI_REQUESTS** macro into the line or phone class override.

```
class CJTLine : public CTSPILineConnection
{
// Overrides from CTSPILineConnection
public:
        CJTDevice* GetDeviceInfo() { return (CJTDevice*) CTSPIConnection::GetDeviceInfo(); }
        virtual std::istream& read( std::istream& istm );
protected:
        virtual DWORD OnLineFeaturesChanged(DWORD dwLineFeatures);
        virtual bool UnsolicitedEvent(LPCVOID lpBuff);
        virtual void OnTimer();
        // Create the event map
        DECLARE_TSPI_REQUESTS()
        // TSPI handlers
        bool OnAnswer(CTSPIRequest* pReq, LPCVOID lpBuff);
        bool OnDropCall(CTSPIRequest* pReq, LPCVOID lpBuff);
        bool OnMakeCall(CTSPIRequest* pReq, LPCVOID lpBuff);
};
```

Then in one of the source files, declare the route map that defines where each request should be sent. There should be a route map definition for each **DECLARE_TSPI_REQUESTS** macro used.

```
/*-----------------------------------------------------------------------------*/
// TSPI Request map
/*-----------------------------------------------------------------------------*/
BEGIN_TSPI_REQUEST(CJTLine)
        ON_AUTO_TSPI_REQUEST(REQUEST_ACCEPT)
        ON_TSPI_ REQUEST_ANSWER(OnAnswer)
        ON_TSPI_REQUEST _MAKECALL(OnMakeCall)
        ON_TSPI_REQUEST(REQUEST_DROPCALL, OnDropCall)
END_TSPI_REQUEST()
```

In the above example,

- The **TSPI_lineAccept** function would be completed with a zero return code without any processing by this service provider (other than what TSP++ does automatically which is to change the call-state of the call to **LINECALLSTATE_ACCEPTED**).
- The **TSPI_lineAnswer** request would call **CJTLine::OnAnswer** and pass it the **RTAnswer** request which defines the TAPI command. Note that the recipient function must be declared as **bool OnAnswer(RTAnswer*** *pRequest***, LPVOID** *lpBuff***);**
- The **TSPI_lineMakeCall** request would call **CJTLine::OnMakeCall** and pass it the **RTMakeCall** request which defines the TAPI command. Note that the recipient function must be declared as **bool OnMakeCall(RTMakeCall*** *pRequest***, LPVOID** *lpBuff***);**

✍✍The **TSPI_lineDropCall** request would call **CJTLine::OnDropCall** and pass it the **RTDropCall** request which defines the TAPI command. The request object would be passed as a pointer to a **CTSPIRequest** object rather than the derived **RTDropCall** type. This allows for the same member function to process more than one request type through the **ON_TSPI_REQUST** macro.

The **ReceiveData** method is managed internally in the library (although it can be overridden if you want to change the functionality) and uses the created map to determine where each request should be sent for a line or phone device.

If the request is not defined in the map, then it is completed automatically with a **LINEERR_OPERATIONUNAVAIL** error code.

DefinedRequests

Depending on the features available in the hardware, different functions will need to be implemented by the service provider. The following chart shows some of the relationships between request identifiers used in the **ON_TSPI_REQUEST** macro and the operation.

It is important to remember that this is the list of requests that *an application* can perform with the service provider. So, the device might have a display the switch changes, but that doesn't mean that **TSPI_phoneSetDisplay** should be exported unless the application can change the display as well.

Common operations that *retrieve* information from the TSP (such as **TSPI_phoneGetDisplay**) do not have an asynchronous request identifier since they are implemented completely within the TSP++ library.

Note that the appropriate TSPI function listed below must be exported in order to process the command.

Feature	Required Export Function	TSP++ Request Identifier
Make a call	TSPI_lineMakeCall	REQUEST_MAKECALL
Dial on an existing call	TSPI_lineDial	REQUEST_DIAL
Drop a call	TSPI_lineDrop	REQUEST_DROPCALL
Answer an incoming call	TSPI_lineAnswer	REQUEST_ANSWER
Accept an offering call	TSPI_lineAccept	REQUEST_ACCEPT
Place a call on hold	TSPI_lineHold	REQUEST_HOLD
Swap between consultation and holding call	TSPI_lineSwapHold	REQUEST_SWAPHOLD
Take a call off hold	TSPI_lineUnhold	REQUEST_UNHOLD
Transfer a call unsupervised	TSPI_lineBlindTransfer	REQUEST_BLINDXFER

unsupervised		
Transfer a call supervised	TSPI_lineSetupTransfer TSPI_lineCompleteTransfer	REQUEST_SETUPXFER REQUEST_COMPLETEXFER
Issue a call completion	TSPI_lineCompleteCall	REQUEST_COMPLETECALL
Cancel a call completion	TSPI_lineUncompleteCall	REQUEST_UNCOMPLETECALL
Forward a line/address	TSPI_lineForward	REQUEST_FORWARD
Park a call	TSPI_linePark	REQUEST_PARK
Unpark a call	TSPI_lineUnpark	REQUEST_UNPARK
Pickup a call from other station	TSPI_linePickup	REQUEST_PICKUP
Redirect a call to other station	TSPI_lineRedirect	REQUEST_REDIRECT
Secure a call from interferance	TSPI_lineSecure	REQUEST_SECURECALL
Send message using phone display or other notification	TSPI_lineSendUserToUser	REQUEST_SENDUSERINFO
Change the terminal associated with an extension	TSPI_lineSetTerminal	REQUEST_SETTERMINAL
Monitor standard DTMF pulses on the call	TSPI_lineMonitorDigits	Call OnDigit when digits are seen on the call
Monitor Media changes on the line	TSPI_lineMonitorMedia	Call OnDetectedNewMediaModes when media changes are seen on the call.
Monitor tones generated on the call	TSPI_lineMonitorTones	Call OnTone when media changes are seen on the call
Gather incoming digits on the call	TSPI_lineGatherDigits	Call OnDigit when digits are seen on the call
Generate DTMF or pulse digits on a connected call	TSPI_lineGenerateDigits	REQUEST_GENERATEDIGITS
Generate custom tones on a connected call	TSPI_lineGenerateTone	REQUEST_GENERATETONE
Begin building a conference call with an	TSPI_lineSetupConference	REQUEST_SETUPCONF

conference call with an existing single party call		
Prepare to add a call to the conference (create a consultation call)	TSPI_linePrepareAddTo Conference	REQUEST_PREPAREADDCONF
Add a call to the conference	TSPI_lineAddToConference	REQUEST_ADDCONF
Remove a call from the conference	TSPI_lineRemoveFrom Conference	REQUEST_REMOVECONF
Set call parameters such as current data rate for ISDN	TSPI_lineSetCallParameters	REQUEST_SETCALLPARAMS
Adjust the media stream for a call	TSPI_lineSetMediaControl	REQUEST_MEDIACONTROL
Release user-user information received for a call	TSPI_lineReleaseUserUserInfo	REQUEST_RELEASEUSERINFO – note this is implemented complety within TSP++, simply complete the request with a zero return code.
Associate application information with a call (call data)	TSPI_lineSetCallData	REQUEST_SETCALLDATA – note this is implemented complety within TSP++, simply complete the request with a zero return code.
Set the quality of service for the line (IP Telephony)	TSPI_lineSetQualityOfService	REQUEST_SETQOS
Set the call treatment (silence, ringback, music, etc.)	TSPI_lineSetCallTreatment	REQUEST_SETCALLTREATMENT
Adjust the current state of the line device through an application	TSPI_lineSetLineDevStatus	REQUEST_SETDEVSTATUS
Set the current set of groups an agent is associated with – also used to logon/logoff in some environments.	lineSetAgentGroup	REQUEST_SETAGENTGROUP
Set the current agent state for an address	lineSetAgentState	REQUEST_SETAGENTSTATE
Set the current agent activity	lineSetAgentActivity	REQUEST_SETAGENTACTIVITY

Perform device-specific agent management	lineAgentSpecific	REQUEST_AGENTSPECIFIC
Set the state or properties of a button on the phone device	TSPI_phoneSetButtonInfo	REQUEST_SETBUTTONINFO
Set the text in the phone display	TSPI_phoneSetDisplay	REQUEST_SETDISPLAY
Set the gain of a speaker device on the phone (could be handset, headset, or speakerphone device)	TSPI_phoneSetGain	REQUEST_SETHOOKSWITCHGAIN
Set the hookswitch state for a phone device	TSPI_phoneSetHookswitch	REQUEST_SETHOOKSWITCH
Set the lamp state (steady, blinking, etc.) for a lamp on the phone device.	TSPI_phoneSetLamp	REQUEST_SETLAMP
Set the ring pattern and volume for the phone device.	TSPI_phoneSetRing	REQUEST_SETRING
Set the speaker volume of a phone device.	TSPI_phoneSetVolume	REQUEST_SETHOOKSWITCHVOL
Set the contents of an internal buffer on the phone device	TSPI_phoneSetData	REQUEST_SETPHONEDATA
Get the contents of an internal buffer on the phone device.	TSPI_phoneGetData	REQUEST_GETPHONEDATA

Generated request objects

TSP++ defines specific objects for each asynchronous request passed from TAPI. These objects are then passed to the worker function defined in the request map. This table shows the object which will be passed for each given request type:
For more information on the created objects, refer to the reference sections on each specific object later in this document.

TSP++ Request Macro used	Object passed to worker function
ON_TSPI_REQUEST	CTSPIRequest (base class) – could be *any* request type.
ON_TSPI_REQUEST_ACCEPT	RTAccept
ON_TSPI_REQUEST_ADDCONF	RTAddToConference

ON_TSPI_REQUEST_ANSWER	RTAnswer
ON_TSPI_REQUEST_BLINDXFER	RTBlindTransfer
ON_TSPI_REQUEST_COMPLETECALL	RTCompleteCall
ON_TSPI_REQUEST_COMPLETEXFER	RTCompleteTransfer
ON_TSPI_REQUEST_DIAL	RTDial
ON_TSPI_REQUEST_DROPCALL	RTDropCall
ON_TSPI_REQUEST_FORWARD	RTForward
ON_TSPI_REQUEST_HOLD	RTHold
ON_TSPI_REQUEST_MAKECALL	RTMakeCall
ON_TSPI_REQUEST_PARK	RTPark
ON_TSPI_REQUEST_PICKUP	RTPickup
ON_TSPI_REQUEST_REDIRECT	RTRedirect
ON_TSPI_REQUEST_REMOVEFROMCONF	RTRemoveFromConference
ON_TSPI_REQUEST_SECURECALL	RTSecureCall
ON_TSPI_REQUEST_SENDUSERINFO	RTSendUserInfo
ON_TSPI_REQUEST_SETCALLPARAMS	RTSetCallParams
ON_TSPI_REQUEST_SETTERMINAL	RTSetTerminal
ON_TSPI_REQUEST_SETUPCONF	RTSetupConference
ON_TSPI_REQUEST_SETUPXFER	RTSetupTransfer
ON_TSPI_REQUEST_SWAPHOLD	RTSwapHold
ON_TSPI_REQUEST_UNCOMPLETECALL	RTUncompleteCall
ON_TSPI_REQUEST_UNHOLD	RTUnhold
ON_TSPI_REQUEST_UNPARK	RTUnpark
ON_TSPI_REQUEST_MEDIACONTROL	RTSetMediaControl
ON_TSPI_REQUEST_PREPAREADDCONF	RTPrepareAddToConference
ON_TSPI_REQUEST_GENERATEDIGITS	RTGenerateDigits
ON_TSPI_REQUEST_GENERATETONE	RTGenerateTone
ON_TSPI_REQUEST_RELEASEUSERINFO	RTReleaseUserInfo
ON_TSPI_REQUEST_SETCALLDATA	RTSetCallData
ON_TSPI_REQUEST_SETQOS	RTSetQualityOfService
ON_TSPI_REQUEST_SETCALLTREATMENT	RTSetCallTreatment
ON_TSPI_REQUEST_SETDEVSTATUS	RTSetLineDevStatus
ON_TSPI_REQUEST_SETBUTTONINFO	RTSetButtonInfo
ON_TSPI_REQUEST_SETDISPLAY	RTSetDisplay
ON_TSPI_REQUEST_SETHOOKSWITCHGAIN	RTSetGain
ON_TSPI_REQUEST_SETHOOKSWITCH	RTSetHookswitch
ON_TSPI_REQUEST_SETLAMP	RTSetLampInfo
ON_TSPI_REQUEST_SETRING	RTSetRing
ON_TSPI_REQUEST_SETHOOKSWITCHVOL	RTSetVolume
ON_TSPI_REQUEST_SETPHONEDATA	RTSetPhoneData
ON_TSPI_REQUEST_GETPHONEDATA	RTGetPhoneData
ON_TSPI_REQUEST_SETAGENTGROUP	RTSetAgentGroup
ON_TSPI_REQUEST_SETAGENTSTATE	RTSetAgentState
ON_TSPI_REQUEST_SETAGENTACTIVITY	RTSetAgentActivity

ON_TSPI_REQUEST_AGENTSPECIFIC	RTAgentSpecific

Constructor and Destructor

CTSPIRequest	Constructor for the request object
~CTSPIRequest	Destructor for the request object. This should be overridden if you define your own request types.

Operations - Public Methods

EnterState	Atomic **SetState** method which checks the current state first. This should be used if multiple threads can access the request at that point in the code.
GetAddressInfo	Returns the **CTSPIAddressInfo** object associated with this request.
GetAsynchRequestId	Returns the TAPI generated asynchronous request ID.
GetCallInfo	Returns the **CTSPICallAppearance** (if any) associated with this request.
GetCommand	Returns the **REQUEST_xxx** value for this command
GetConnectionInfo	Returns the **CTSPILineConnection** or **CTSPIPhoneConnection** associated with this request.
GetLineOwner	Returns the **CTSPILineConnection** that is associated with this request.
GetPhoneOwner	Returns the **CTSPIPhoneConnection** that is associated with this request.
GetRequestName	Returns the textual name of the request object for log purposes.
GetState	Returns the current state of this request (service provider defined).
GetStateTime	Returns the tick count when the request object entered the current state.
HaveSentResponse	Return TRUE/FALSE indicating whether TAPI has been notified about this request finishing.
SetState	Changes the current state of this request (service provider defined)
WaitForCompletion	Wait for the request to finish.

Operations - Protected Methods

Init	Called directly after the constructor by the **CTSPIDevice** owner object
Complete	Completes the request and potentially notifies TAPI.
UnblockThreads	Releases all threads waiting on this request to complete.

Overridables - Protected Methods

Failed	Called if the request is completed with a non-zero return code.

CTSPIRequest::Complete

Protected

virtual void Complete (LONG *lResult* **= 0L, bool** *fSendTN* **= TRUE);**

lResult	Final result code for this request.
fSendTN	Whether to notify TAPI that the request is complete.

Remarks

This method is called by the device object when the request is completed. It optionally notifies TAPI that the request is complete, and runs through the line/phone owner, address owner, and associated call and notifies each object that the request is complete.

Any waiting thread on the request is released. A thread can wait on a request using the **WaitForCompletion** method.

Generally, you should call the **CTSPIConnection** object to complete requests so that the request is removed from the asynchronous request list.

CTSPIRequest::CTSPIRequest

CTSPIRequest(LPCTSTR *pszType***);**
CTSPIRequest(const CTSPIRequest& *src***);**

pszType	Textual Name of this request
src	Existing **CTSPIRequest** to copy

Remarks

Default and copy constructor for the request object. These should only be called by the TSP++ library code.

CTSPIRequest::~CTSPIRequest

virtual ~CTSPIRequest();

Remarks

Destructor for the request object. Should be overridden by any derived request object.

CTSPIRequest::EnterState

bool EnterState(int *iLookForState***, int** *iNextState***);**

iLookForState	**STATE_xxx** state to look for
iNextState	**STATE_xxx** to change to.

Remarks

This method is an atomic operation to change the state of the request object (similar to the **SetState** method).

If the current request state matches *iLookForState*, then the request state is changed to *iNextState*.

Return Value

TRUE if the request state was changed to *iNextState*. FALSE if the request state was not changed or was already *iNextState*.

CTSPIRequest::Failed

protected

virtual void Failed(LONG *lResult***);**

lResult	Final return code for this request

Remarks

This method is called by TSP++ when the request is completed with a non-zero return code. It is used by derived objects to perform automatic *cleanup* for the request.

CTSPIRequest::GetAddressInfo

CTSPIAddressInfo* GetAddressInfo() const;

Remarks

This method returns the associated **CTSPIAddressInfo** that this request is working with.

Return Value

The address object the request is associated to or NULL if there is no object.

CTSPIRequest::GetAsynchRequestId

DRV_REQUESTID GetAsynchRequestId() const;

Remarks

This method returns the TAPI asynchronous request identifier that is associated to this running request.

Return Value

The 32-bit asynchronous request identifier assigned by TAPI when it started the request.

CTSPIRequest::GetCallInfo

CTSPICallAppearance* GetCallInfo() const;

Remarks

This method returns the associated **CTSPICallAppearance** that this request is working with.

Return Value

The call object the request is associated to. NULL if there is no call.

CTSPIRequest::GetCommand

int GetCommand() const;

Remarks

This method returns the **REQUEST_xxx** command that this request object represents.

Return Value

The defined request type (see the above section on *Processing a Request* for more information).

CTSPIRequest::GetConnectionInfo

CTSPIConnectionInfo* GetConnectionInfo() const;

Remarks

This method returns the associated **CTSPILineConnection** or **CTSPIPhoneConnection** that this request is working with.

If the owner type (line or phone) is known then it is preferable to use the **GetLineOwner** or **GetPhoneOwner** methods instead of this method.

Return Value

The line/phone connection object the request is associated to or NULL if there is no object.

CTSPIRequest::GetLineOwner

CTSPILineConnectionInfo* GetLineOwner() const;

Remarks

This method returns the associated **CTSPILineConnection** that this request is working with.

Return Value

The line connection object the request is associated to or NULL if there is no object.

CTSPIRequest::GetPhoneOwner

CTSPIPhoneConnectionInfo* GetPhoneOwner() const;

Remarks

This method returns the associated **CTSPIPhoneConnection** that this request is working with.

Return Value

The phone connection object the request is associated to or NULL if there is no object.

CTSPIRequest::GetRequestName

LPCTSTR GetRequestName() const;

Remarks

This method returns the associated request name that was assigned to the request constructor.

Return Value

A pointer to a textual name.

CTSPIRequest::GetState

int GetState() const;

Remarks

This method returns the current assigned state of the request packet. The only library-defined state is **STATE_INITIAL** that is the first state the request is placed into. This is what drives the state-machine mechanics of the provider model.

Return Value

The request state in effect for this object.

CTSPIRequest::GetTime

DWORD GetStateTime() const;

Remarks

This method returns the machine tick count at the time that the request packet entered into the current state (either by **SetState** or **EnterState**). It may be used to determine how long the request has been processing the current step.

Return Value

The value of **GetTickCount** when the request entered the current state.

CTSPIRequest::HaveSentResponse

bool HaveSentResponse() const;

Remarks

This method returns whether or not this request has replied to TAPI as to the final result code. There are times when TAPI must be told the request is completed before the request is actually done processing. In these cases, the **CompleteRequest** method of the **CTSPILineConnection** or **CTSPIPhoneConnection** can be told to *not* delete the request, but reply to TAPI anyway.

Return Value

TRUE if TAPI has been told the request finished.

FALSE if TAPI has not yet been notified about the request completion.

CTSPIRequest::Init

Protected

virtual void Init(CTSPILineConnection* *pConn***, CTSPIAddressInfo*** *pAddr***,**
CTSPICallAppearance* *pCall***, int** *iRequest***,**
DRV_REQUESTID *dwRequestId***);**
void Init(CTSPIPhoneConnection* *pConn***, int** *iRequest***,**
DRV_REQUESTID *dwRequestId***);**

pConn	Phone or line connection object which is associated with request.
pAddr	Address object which is associated with request
pCall	Call object which is associated with request
iRequest	Command type (**REQUEST_xxx**)
dwRequestId	TAPI asynchronous request id which identifies this request.

Remarks

This method is called right after the constructor to initialize the request object. This is called *before* the request is inserted into the device request list. In general, you should never need to override this method.

CTSPIRequest::SetState

void SetState(int *iState***) const;**

iState	State to transition the request into.

Remarks

This method sets the current assigned state of the request packet. The only library-defined state is **STATE_INITIAL** that is the first state the request is placed into. This is what drives the state-machine mechanics of the provider model.

The time that the request entered the given state is recorded by TSP++ and may be retrieved using the **GetStateTime** method. This may be used for request time-outs.

CTSPIRequest::UnblockThreads

void UnblockThreads();

Remarks

This method releases any threads that are waiting on the request to complete. It is used by TSP++ when the request is completed

CTSPIRequest::WaitForCompletion

virtual LONG WaitForCompletion (DWORD *dwMSecs***);**

dwMSecs	Number of milliseconds to wait for the request to complete, **INIFINITE** to wait forever (not recommended).

Remarks

This method causes the current thread to be put to sleep waiting for the request object to be completed. The thread will be woken either when the request is complete, or when the elapsed wait time is over.

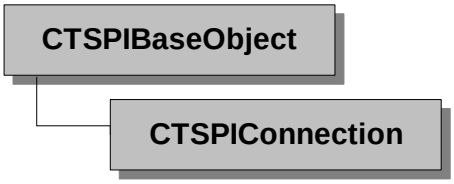
Warning: Do not put the processing thread to sleep unless you are certain that the request will complete!

Return Value

Final TAPI result code of the request.

(-1) if the request timed-out without completing within the given **dwMSecs** time period.

CTSPIConnection



The **CTSPIConnection** is a base class defined in the library to represent a generic connection to a physical medium. It is the base class for the line connection (**CTSPILineConnection**), and phone connection (**CTSPIPhoneConnection**) objects. It cannot be used on its own (it is a virtual base class), and is only documented for the methods which are usable from the line and phone devices.

Asynchronous Request List

The connection object is responsible for maintenance of all the requests pending on that line or phone device. Most requests which are generated by TAPI are assigned an asynchronous request identifier which tags the request, so that when it is completed, the service provider can inform TAPI as to the result of the operation (for information on this process, see the section on **CTSPIRequest**). The service provider is required to initiate the operation and then return the asynchronous identifier to TAPI that will allow the application to proceed with other tasks until the operation completes.

Since there may be multiple pending requests for a line or phone device, each request from the TAPI server which is asynchronous, will generate an asynchronous request object and be stored into a list maintained by the connection. This request list is a FIFO queue, with the topmost request being the one currently being executed. As each request completes, the connection object will remove the request object from the list and inform TAPI about the final result of the operation.

If the service provider has asynchronous requests that stay in the queue for long periods of time, it may be necessary to override the **AddAsynchRequest** method in order to allow other requests to start when the current request has not yet finished. This method is called internally to add each request to the asynchronous list and returns whether or not to immediately activate the request.

Connection Opens and Closes

Each time a line or phone device receives an open request from TAPI through the **lineOpen** or **phoneOpen** API, it calls the **OpenDevice** method of this base connection object. When the line or phone is closed, the **CloseDevice** method will be invoked. The default behavior of these two methods is to call the device object **Open/Close** methods which will eventually send a request to the companion application using the **CServiceProvider::OpenDevice** and **CServiceProvider::CloseDevice** methods.

One of the open/close methods must be overridden by 32-bit providers to actually provide support for the opening and closing of the device.

Generally, it is recommended that this is handled in the Device Object (see **CTSPIDevice**).

Synchronous requests

In some cases, it is advantageous or even necessary to wait for a request to finish. The virtual method **WaitForRequest** will wait a specified amount of time for the request to finish. Any request actually passed by TAPI, unless noted, should not be waited for. This functionality should be used for internal commands only. In most cases, the synchronous requests from the TAPI server are handled without any interaction by the derived provider.

Constructor and Destructor

CTSPIConnection	Internal constructor for the connection object
~CTSPIConnection	Destructor for the connection object.

Operations - Public Members

AddAsynchRequest	Create a new asynchronous request and add it to our list. Start the request if no active request is pending.
AddDeviceClass	Adds a new device class informational structure to the line/phone DEVCAPS structure.
CompleteCurrentRequest	Complete the current head request. This can optionally remove and delete the request.
CompleteRequest	Complete a specific request and notify TAPI. This can optionally remove and delete the request.
FindRequest	Find a request based on some criteria.
GetConnInfo	Return the connection (switch or phone) information for this connection. This will be reported in the **LINEDEVCAPS** and **PHONEDEVCAPS** automatically.
GetDeviceClass	Returns a **DEVICECLASS** object from a class name.
GetDeviceID	Returns the TAPI-assigned device ID for this line or phone.
GetDeviceInfo	Returns a pointer to the **CTSPIDevice** owner of this connection object.
GetName	Return the textual name assigned to this phone or line device.
GetNegotiatedVersion	Returns the negotiated TSPI version for this phone or line device. This will be the highest version used for reporting line or phone events while the device is open.
GetRequest	Return a request based on position.
GetRequestCount	Return the count of requests pending.
HasBeenDeleted	Returns whether this device has been dynamically

	removed by the service provider.
IsLineDevice	Returns TRUE/FALSE based on whether the connection is a line device. Mutually exclusive from the **IsPhoneDevice** method.
IsPhoneDevice	Returns TRUE/FALSE based on whether the connection is a phone device. Mutually exclusive from the **IsLineDevice** method.
RemoveDeviceClass	Removes an associated device class structure from the provider list.
RemovePendingRequests	This removes all pending requests based on a criteria.
RemoveRequest	Remove a request from our list. It can optionally be deleted.
SetExtVersionInfo	Initializes the extension information for the service provider.
SetName	This member method allows the name of the device to be set. Since the object is constructed in the library, the first time the derived service provider can change the information is in **CServiceProvider::providerInit**, and the device has already been setup.
WaitForAllRequests	Waits for all pending requests of the type specified on this line or phone device.

Static Functions

ReadString	Function to read a string from an iostream buffer.

Overridables - Public Members

DispatchRequest	This function is used to route the TAPI requests through the TSPI request map.
CloseDevice	Called when the last line/phone closes the device. The default implementation calls the **CServiceProvider::CloseDevice** method.
GetCurrentRequest	Return the current head of the request list.
GetExtensionID	Called in response to **TSPI_lineGetExtensionID** or **TSPI_phoneGetExtensionID**.
GetExtVersion	Returns the currently selected extension version.
GetIcon	Called in response to **TSPI_lineGetIcon** for this device.
GetID	This is called in response to a **lineGetID** or **phoneGetID** request when a line or phone object was the target. It returns resource handle and device information about the device based on a passed device class.
NegotiateVersion	Called in response to **TSPI_lineNegotiateAPIVersion** or **TSPI_phoneNegotiateAPIVersion** for this device.
NegotiateExtVersion	Called in response to **TSPI_lineNegotiateExtVersion** or

	TSPI_phoneNegotiateExtVersion for this device.
OpenDevice	Called when the first line/phone opens the device. The default implementation calls the **CServiceProvider::OpenDevice** method.
ReceiveData	Called by the **CTSPIDevice::ReceiveData** method when data is received from the companion application for this specific connection (or for any line/phone on the device). Default implementation calls the **CServiceProvider::ProcessData** method.
SelectExtVersion	Called in response to **TSPI_lineSelectExtVersion** or **TSPI_phoneSelectExtVersion** for this device.
UnsolicitedEvent	Called by the **ReceiveData** method if no worker function takes responsibility for an event.
WaitForRequest	Wait for a request to finish with time-out.

Overridables - Protected Members

AddAsynchRequest	Adds the request to the internal request list and returns whether or not to immediately start the request.
GetRequestList	Internal function which returns a pointer to the macro-generated request handler list.
GetRequestMap	Internal function which returns the request map which is used to route TAPI requests for this connection. See the comparable function in the derived classes.
Init	Called directly after the constructor by the **CServiceProvider** owner object
OnCancelRequest	This is called when a request is canceled on the connection.
OnNewRequest	Processes a new request for the connection
OnRequestComplete	This is called when a request completes on this connection.
OnTimer	This method is called in response to the periodic timer generated by the **CTSPIDevice** object. It may be overridden to process events.

Operations - Protected Members

SetDeviceID	This method is called by the **CServiceProvider** class when a new line/phone is created and added to the system. It changes the temporary device id assigned by **CTSPIDevice::CreateLine** and **CTSPIDevice::CreatePhone** to a device id defined by TAPI.

CTSPIConnection::AddAsynchRequest

int AddAsynchRequest(CTSPIRequest* *pRequest***);**

pRequest	Request object to add to the request list

Remarks

This method inserts a new request into the pending request list. If there are no running requests in the list, then this request is actually started and given the opportunity to run the **STATE_INITIAL** processing before this method returns.

If the derived provider wants to change the position where this request is inserted into the list, override the **OnNewRequest** method in either the line, phone, or device objects.

Return Value

Index position where this request was inserted into the asynchronous list.

CTSPIConnection::AddAsynchRequest

Protected

virtual bool AddAsynchRequest (CTSPIRequest* *pReq***, int** *iPos***);**

pReq	Request object to insert into the request list.
iPos	Position to insert the request into (-1 = end of list).

Remarks

This method inserts a new request into the pending request list. If there are no running requests in the list, then this request is actually started and given the opportunity to run the **STATE_INITIAL** processing before this method returns.

This is the method called by the public versions of **AddAsynchRequest**. It is documented specifically so that it may be overridden if positional information is being ignored.

Return Value

TRUE if the request was inserted into the list.

FALSE if the request was not inserted and will not be run.

CTSPIConnection::AddDeviceClass

int AddDeviceClass (LPCTSTR *pszClass***, DWORD** *dwData***);**

int AddDeviceClass (LPCTSTR *pszClass***, HANDLE** *hHandle***,**
LPCTSTR *lpszBuff***);**

int AddDeviceClass (LPCTSTR *pszClass***, HANDLE** *hHandle***, LPVOID** *lpBuff***,**
DWORD *dwSize***);**

int AddDeviceClass (LPCTSTR *pszClass*, **DWORD** *dwFormat*, **LPVOID** *lpBuff*,
DWORD *dwSize*, **HANDLE** *hHandle* = **INVALID_HANDLE_VALUE);**

int AddDeviceClass (LPCTSTR *pszClass*, **LPCTSTR** *pszBuff*,
DWORD *dwType* = **-1L);**

pszClass	Device class to add/update for (e.g. "tapi/line", etc.)
dwData	DWORD data value to associate with device class.
hHandle	Win32 handle to associate with device class.
lpszBuff	Null-terminated string to associate with device class.
lpBuff	Binary data block to associate with device class
dwSize	Size of the binary data block
dwType	**STRINFORMAT** of the **lpszBuff** parameter.

Remarks

This method adds an entry to the device class list, associating it with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEDEVCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

Index array of added structure.

CTSPIConnection::CloseDevice

virtual bool CloseDevice();

Remarks

This method is called when **TSPI_lineClose** or **TSPI_phoneClose** is called by TAPI.

The default behavior is to call the **CTSPIDevice::CloseDevice** method for processing.

Return Value

TRUE if the connection device closed.

FALSE if the close failed.

CTSPIConnection::CompleteCurrentRequest

bool CompleteCurrentRequest(LONG *lResult* = **0, bool** *fTellTAPI* = **TRUE,**
bool *fRemove* = **TRUE);**

lResult	Final result code for the head request in our list.
fTellTAPI	TRUE if TAPI should be notified that the request is complete.
fRemove	TRUE if we should delete the request from the list.

Remarks

This method locates the current running request (the first request in the asynchronous list) and calls the **CTSPIRequest::Complete** method to complete the request. It then removes the request from the list if the **fRemove** parameter is **TRUE**.

Return Value

TRUE if the request was completed or updated with new information. FALSE if the completion failed.

CTSPIConnection::CompleteRequest

bool CompleteRequest(CTSPIRequest* *pReq*, **LONG** *lResult* **= 0,**
bool *fTellTAPI* **= TRUE, bool** *fRemove* **= TRUE);**

pReq	Request object to complete
lResult	Final result code for the head request in our list.
fTellTAPI	TRUE if TAPI should be notified that the request is complete.
fRemove	TRUE if we should delete the request from the list.

Remarks

This method calls the **CTSPIRequest::Complete** method to complete the request. It then removes the request from the list if the **fRemove** parameter is **TRUE**.

Return Value

TRUE if the request was completed or updated with new information. FALSE if the completion failed.

CTSPIConnection::DispatchRequest

virtual bool DispatchRequest(CTSPIRequest* *pRequest*, **LPCVOID** *lpBuff*);

pRequest	Request object to process.
lpBuff	Optional buffer for the inbound hardware event.

Remarks

This method searches the request map and dispatches the given request and associated device event to the approprite handler for the line or phone object. If no handler is found, the request is automatically completed.

Return Value

Boolean indicator whether request was processed by an event handler.

CTSPIConnection::FindRequest

CTSPIRequest* FindRequest(CTSPICallAppearance* pCall,
int iReqType);

pCall	Call object associated with request (NULL for any)
iReqType	**REQUEST_xxx** key to locate.

Remarks

This method searches the request list looking for the specified request. It returns the first located request.

Return Value

Located **CTSPIRequest** object or NULL if no request matching the criteria was found.

CTSPIConnection::GetCurrentRequest

virtual CTSPIRequest* GetCurrentRequest(BOOL *fAddRef* **= FALSE) const;**

fAddRef	Whether to increment the reference count on the request.

Remarks

This method returns the current request that is being processed by the line or phone device. The default behavior is to return the first request in the array.

Return Value

Pointer to the current request.

CTSPIConnection::GetDeviceClass

DEVICECLASSINFO* GetDeviceClass(LPCTSTR *pszClass***);**

pszClass	Device class to search for (e.g. "tapi/line", etc.)

Remarks

This method returns the device class structure associated with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEDEVCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

DEVICECLASSINFO structure that represents the specified text name or NULL if no association has been performed.

CTSPIConnection::GetDeviceID

DWORD GetDeviceID() const;

Remarks

This method returns the device identifier associated with this device. This is a combination of the permanent provider id associated to the parent **CTSPIDevice** object, and the index into the array where this line or phone is.

This device identifier is used to associate a command to the line or phone with the companion applications in the 16-bit library.

Return Value

Unique identifier to the line or phone within the TSP.

CTSPIConnection::GetDeviceInfo

CTSPIDevice* GetDeviceInfo() const;

Remarks

This method returns the device parent for this line or phone device.

Return Value

Parent **CTSPIDevice** object for the line or phone.

CTSPIConnection::GetExtensionID

virtual LONG GetExtensionID(DWORD *dwTSPIVersion***,**
LPEXTENSIONID *lpExtensionID***);**

dwTSPIVersion	Previously negotiated TSPI version.
lpExtensionID	Pointer to **EXTENSIONID** block to fill in.

Remarks

This method is called when TAPI invokes the **TSPI_lineGetExtensionID** or **TSPI_phoneGetExtensionID** function. It returns device-specific extension information.

The default implementation returns the information given to the **SetExtVersionInfo** method. If no extended version information has been set, the method returns **LINEERR_OPERATIONUNAVAIL**.

Return Value

TAPI Result code

CTSPIConnection::GetExtVersion

DWORD GetExtVersion() const;

Remarks

This method returns the currently selected extension version (set by **TSPI_lineSelectExtVersion** or **TSPI_phoneSelectExtVersion**) for this line or phone device.

Return Value

Selected extension version, zero if no version has been selected.

CTSPIConnection::GetIcon

virtual LONG GetIcon (TString& *strDevClass***, LPHICON** *lphIcon***);**

strDevClass	Device class to retrieve icon for (“tapi/line”, etc.)
lphIcon	Returning icon handle

Remarks

This method returns an icon for the requested device class. It is called in response to the **TSPI_lineGetIcon** function.

The default implementation returns **LINEERR_OPERATIONUNAVAIL**. TAPI will automatically return an icon for the request if it is failed by the service provider with this error code.

If you need to return a specific icon for the line/phone devices, override this method in the line or phone class and fill in the *lphIcon* pointer with the correct icon.

Return Value

TAPI result code

CTSPIConnection::GetID

virtual LONG GetID (TString& *strDevClass***, LPVARSTRING** *lpDeviceID***, HANDLE** *hTargetProcess***);**

strDeviceClass	Device class which is being asked for.
lpDeviceID	Returning device information.
hTargetProcess	Process requesting data.

Remarks

This method is called in response to an application calling **lineGetID** or **phoneGetID**. It is used to return device information to the application based on a string device class key. TSP++ implements this method internally and returns any device information which was added using **CTSPIConnection::AddDeviceClass**.

Any handle that was given to the **AddDeviceClass** method is automatically duplicated in the given process for TAPI 2.x using the **DuplicateHandle** Win32 function.

Return Value

TAPI error code or zero if the method was successful.

CTSPIConnection::GetName

LPCTSTR GetName() const;

Remarks

This method returns the name of the line or phone device that was assigned during the creation of the object.

Return Value

String pointer representing ASCII or UNICODE name of line/phone device.

CTSPIConnection::GetNegotiatedVersion

DWORD GetNegotiatedVersion() const;

Remarks

This method returns the negotiated TAPI version decided on when the line or phone was initially opened by TAPI.

Return Value

TAPI version negotiated by TAPI system components and this TSP.

CTSPIConnection::GetRequest

CTSPIRequest* GetRequest(int *iPos*, **BOOL** *fAddRef* **= FALSE) const;**

iPos	Index in the request list for the request being accessed.
fAddRef	Whether to increment the reference count on the request.

Remarks

This method allows random-access direct retrieval of requests within the request list. The passed index should be within the current valid ranges (0 – **GetRequestCount()**-1).

Return Value

Request object at the specified position or NULL if no request was found.

CTSPIConnection::GetRequestCount

int GetRequestCount() const;

Remarks

This method returns the current count of pending requests in the asynchronous request list. This number will be one more than the valid indexes used for **GetRequest**.

Return Value

Current count of pending/active requests.

CTSPIConnection::GetRequestList

Protected

virtual const tsplib_REQMAP* GetRequestList() const;

Remarks

This method returns a pointer to the generated internal request map. This map is declared by the **DECLARE_TSPI_REQUESTS** macro, which also overrides this function for each derived class. This list is then processed during initialization to create a hash lookup map for dispatching requests.

Return Value

Pointer to the global request list.

CTSPIConnection::GetRequestMap

Protected

virtual TRequestMap* GetRequestMap() const = 0;

Remarks

This method should return a pointer to the dynamically created request map which is used to route TAPI requests through the TSP. This method must be supplied by the derived classes.

Return Value

Pointer to the request map.

CTSPIConnection::HasBeenDeleted

bool HasBeenDeleted() const;

Remarks

This method returns whether the given line/phone device has been removed from the valid list of TAPI devices. This happens when the **CTSPIDevice::RemoveLine** or **CTSPIDevice::RemovePhone** member is called for a line/phone device.

Return Value

TRUE if the device is invalid and should not process requests, FALSE if the device is still available.

CTSPIConnection::Init

virtual void Init(CTSPIDevice* *pDeviceOwner*, **DWORD** *dwDeviceID*);

pDeviceOwner	Device owner object for this line/phone.
dwDeviceID	Unique identifier for the line/phone within the TSP.

Remarks

This method initializes the **CTSPIConnection** object and is called directly after the constructor. It is documented here simply for overriding purposes.

CTSPIConnection::IsLineDevice

bool IsLineDevice() const;

Remarks

This method returns whether the **CTSPIConnection** object is representing a line or phone device. Another method of doing this would be through MFCs run-time type information.

Return Value

TRUE if this is a line device.

FALSE if this is a phone device.

CTSPIConnection::IsPhoneDevice

bool IsPhoneDevice() const;

Remarks

This method returns whether the **CTSPIConnection** object is representing a line or phone device. Another method of doing this would be through MFCs run-time type information.

Return Value

TRUE if this is a phone device.

FALSE if this is a line device.

CTSPIConnection::NegotiateExtVersion

virtual LONG NegotiateVersion(DWORD *dwLoVersion***,**
DWORD *dwHiVersion***, LPDWORD** *lpdwExtVersion***);**

dwLoVersion	The low extension version to negotiate to
dwHiVersion	The hi extension version to negotiate to.
lpdwExtVersion	The returning version this line agrees to conform to.

Remarks

This method is called as a result of the TAPI function **TSPI_lineNegotiateExtVersion** or **TSPI_phoneNegotiateExtVersion** against a specific line or phone device.

The default implementation negotiates a valid extension version based on the information given in the **SetExtVersionInfo** method. If no extension information has been set (by the mentioned method) then the method returns an error.

Return Value

TAPI result code

CTSPIConnection::NegotiateVersion

virtual LONG NegotiateVersion(DWORD *dwLoVersion***,**
DWORD *dwHiVersion***, LPDWORD** *lpdwExtVersion***);**

dwLoVersion	The low TAPI version to negotiate to (0x20000000)
dwHiVersion	The hi TAPI version to negotiate to.
lpdwExtVersion	The returning version this line agrees to conform to.

Remarks

This method is called as a result of the TAPI function **TSPI_lineNegotiateAPIVersion** or **TSPI_phoneNegotiateAPIVersion** against a specific line or phone device.

The default implementation negotiates the highest TAPI version as allowed by the version passed in the **CServiceProvider** constructor.

Return Value

TAPI result code

CTSPIConnection::OnCancelRequest

virtual void OnCancelRequest (CTSPIRequest* *pReq***);**

pReq Request object which is about to be canceled.

Remarks

The connection object calls this method when a request is being canceled due to a **RTDropCall** request being inserted into the queue.

The default behavior is to pass control to the **CTSPIDevice::OnCancelRequest** method.

CTSPIConnection::OnNewRequest

virtual bool OnNewRequest (CTSPIRequest* *pReq***);**

pReq Request object which is about to be added to the list.

Remarks

This method is called by the connection object each time a new request packet is added to the request list. It is called before the request is officially added to the list and allows the derived connection object to manipulate the list before insertion.

The default behavior is to pass control to the **CTSPIDevice::OnNewRequest** method.

Return Value

TRUE if the connection object should continue to add the request. FALSE if the request is to be canceled.

CTSPIConnection::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq***, LONG** *lResult***);**

pReq Request object which has completed.
lResult Final result code for request.

Remarks

This method is called when a request completes in our request list. It is derived in the **CTSPILineConnection** and **CTSPIPhoneConnection** objects to cleanup the request once it is completed (or failed).

CTSPIConnection::OnTimer

virtual void OnTimer();

Remarks

This method is called by the **CTSPIDevice** object when an interval timer has been received from either the companion application (16-bit) or interval timer thread (32-bit). It gives the object an opportunity to process idle-time tasks or other cleanup work.

The default behavior is to pass control to the **CServiceProvider::ProcessIntervalTimer** method.

CTSPIConnection::OpenDevice

virtual bool OpenDevice ();

Remarks

This method is called from the **CTSPILineConnection** and **CTSPIPhoneConnection** objects to open the line or phone device.

If the method is not overridden, it will forward the request onto **CTSPIDevice::OpenDevice**.

Return Value

TRUE/FALSE success indicator. If FALSE is returned, the **TSPI_lineOpen** or **TSPI_phoneOpen** that generated the request will fail and return an error.

CTSPIConnection::readString

static void readString(std::istream& *istm***, TString&** *str***);**

istm	Input stream to read from
str	Area to store read string

Remarks

This static function reads a string from the given iostream. It should only be used to read strings written by the SPLUI library code as the format is specific to the TSP++ library.

The *str* variable will contain the string value on return. If there was no data in the stream, *str* will be empty.

CTSPIConnection::ReceiveData

virtual bool ReceiveData (DWORD *dwData***=0, const LPVOID** *lpBuff***=NULL, DWORD** *dwSize***=0);**

dwData	32-bit numeric data response.
lpBuff	Pointer to buffer with received data.
dwSize	Size of the above buffer

Remarks

This method should be called from the **CTSPIDevice** object owner when it receives data from the input thread (32-bit).

If the TSPI routing map mechanism is used then this method does not need to be overridden.

Return Value

TRUE/FALSE success indicator.

CTSPIConnection::RemoveDeviceClass

bool RemoveDeviceClass (LPCTSTR *pszClass*);

pszClass Device class key to remove.

Remarks

This method removes the specified device class information from this object. It will no longer be reported as available to TAPI.

Return Value

TRUE if device class information was located and removed.

CTSPIConnection::RemovePendingRequests

void RemovePendingRequests(CTSPICallAppearance* *pCall* **= NULL,**
int *iReqType* **= REQUEST_ALL,**
LONG *lResult* **= TAPIERR_REQUESTCANCELLED**
CTSPIRequest* *pCurrReq***=NULL);**

pCall	Call object to remove requests for (NULL for all)
iReqType	Request type to locate and remove (**REQUEST_ALL** for all)
lResult	Result code to send to TAPI.
pCurrReq	Pointer to the current request (which is ignored).

Remarks

This method completes all pending requests that match the criteria without processing the request and sends the specified error code back to TAPI. Only requests that match the criteria passed in are processed by this method.

The supplied current request (if specified) is ignored during the deletion.

CTSPIConnection::RemoveRequest

bool RemoveRequest(CTSPIRequest* *pReq***, bool** *fDelete***=FALSE);**

pReq	Request to remove from the list.
fDelete	Whether to delete the request

Remarks

This method completes all pending requests that match the criteria without processing the request and sends the specified error code back to TAPI. Only requests that match the criteria passed in are processed by this method.

Return Value

TRUE/FALSE success indicator.

CTSPIConnection::SelectExtVersion

virtual LONG SelectExtVersion(DWORD *dwExtVersion***);**

dwExtVersion	The version being requested by the owner application.

Remarks

This method is called as a result of the TAPI function **TSPI_lineSelectExtVersion** or **TSPI_phoneSelectExtVersion** against a specific line or phone device.

The default implementation stores off the given version. It can be retrieved from the line or phone using the **GetExtVersion** method.

Return Value

TAPI result code

CTSPIConnection::SetDeviceID

Protected

void SetDeviceID(DWORD *dwID***);**

dwID	New device line/phone id to associate to this object.

Remarks

This method changes the line/phone device id to another value. This is used by the **CServiceProvider** class when a new line or phone is created using the **CTSPIDevice::CreateLine** or **CTSPIDevice::CreatePhone**. Once TAPI finalizes the line or phone addition, it calls the provider back and hands in a new device identifier for the line line/phone.

CTSPIConnection::SetExtVersionInfo

void SetExtVersionInfo(DWORD *dwMinVersion***, DWORD** *dwMaxVersion***,**
DWORD *dwExtensionID0***, DWORD** *dwExtensionID1* **= 0,**
DWORD *dwExtensionID2* **= 0, DWORD** *dwExtensionID3* **= 0);**

dwMinVersion	Minimum extension version (major/minor) to negotiate to.
dwMaxVersion	Maximum extension version (major/minor) to negotiate to.
dwExtensionID0	First Extension ID (used in **TSPI_xxxSelectExtVersion**).
dwExtensionID1	Second Extension ID.
dwExtensionID2	Third Extension ID.
dwExtensionID3	Fourth Extension ID

Remarks

This method sets the extension information for the service provider. This allows the **TSPI_lineSelectExtVersion**, **TSPI_lineNegotiateExtVersion**, **TSPI_phoneSelectExtVersion**, and **TSPI_phoneNegotiateExtVersion** functions to work properly.

In order to provide line/phone version extensions, you must either call this method during service provider initialization *or* override all the extension methods and provide your own implementation.

CTSPIConnection::SetName

void SetName(LPCTSTR *lpszName***);**

lpszName	Name to associate with this line/phone.

Remarks

This method changes the name that is reported back to TAPI in the **PHONECAPS** and **LINECAPS** structures. It should be called during the setup of the line/phone.

CTSPIConnection::UnsolicitedEvent

virtual bool UnsolicitedEvent(LPCVOID *lpBuff***);**

lpBuff	Received event value

Remarks

This method is called by the **ReceiveData** method when no request took ownership of a received event. This can be due to no request currently running, or no request returning a non-zero value when it received this event.

Return Value

The return value ignored by the default **ReceiveData** method but is provided for derivation purposes.

CTSPIConnection::WaitForAllRequests

void WaitForAllRequests(CTSPICallAppearance* *pCall***=NULL,**
int *iRequest***=REQUEST_ALL);**

pCall	Call appearance to look for (NULL indicates ALL)
iRequest	Request id to look for (REQUEST_ALL indicates ALL).

Remarks

This method pauses the calling thread until all requests that match the specified criteria have completed or failed.

Warning: Do not put the processing thread to sleep unless you are certain that all the requests will complete!

CTSPIConnection::WaitForRequest

virtual void WaitForRequest(DWORD *dwTimeout***, CTSPIRequest*** *pReq***);**

dwTimeout	Timeout to wait for the request to complete in milliseconds.

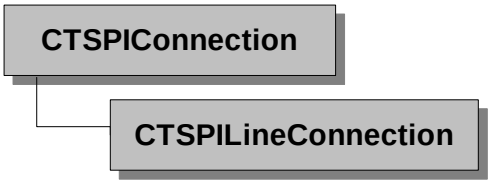
pReq Request to wait for.

Remarks

This method pauses the calling thread until the specified request completes. If the specified timeout occurs, the thread is automatically released even if the request isn't done.

Warning: Do not put the processing thread to sleep unless you are certain that the request will complete!

CTSPILineConnection



The **CTSPILineConnection** object represents a line device in the TSP++ library. It is owned by a **CTSPIDevice** object and manages a set of asynchronous requests that are pending for the line it represents. Each object maintains the state and capabilities of the line connection.

Line Devices

The line device is TAPI's basic representation of a device capable of being controlled by telephony commands. The name *line device* can be applied to many different PBX elements:

1. Any hardware telephony device that is connected to an actual telephone line. This includes things such as fax machines, modems, ISDN cards, and routers.
2. An ACD or PBX queue for holding calls that have not yet been sent to their final destination.
3. A virtual telephone that is manipulated by a computer system to ask and respond to user feedback using DTMF tones.
4. A computer-controlled telephone dialer that has specialized hardware and software to detect voice energy on the far end of the attached call.

Line devices support telephony capabilities by allowing applications to send or receive information to or from a telephone network. TAPI itself views the line as a point of entry that leads to the H/W switch.

Each line device contains a set of one or more addresses that can be used to establish calls. In some cases, a service provider will model a line as having only a single address (POTS), and in others, multiple addresses are required (ISDN or digital network). For more information concerning addresses, see the reference section on *CTSPIAddressInfo*.

Line connection initialization

The line objects are created by the **CTSPIDevice** object during initialization of the service provider or when the service provider calls the **CTSPIDevice::CreateLine** method to dynamically add a line device.

After the line construction is complete, the virtual method **Init** is called to actually initialize the object. This method will initializes the known fields in the **LINEDEVCAPS** and **LINEDEVSTATUS** structures based on what functions the TSP exports and information available in the registry (if the SPLUI library is used).

If the SPLUI library is not used, then the derived service provider is responsible for adding address channels to the line during this initialization process by overriding the object and its **Init** method and calling **CreateAddress** for each channel.

Line Open/Close events

The line is not considered open until a **lineOpen** is received in the **CServiceProvider** class. At this time, the **CTSPILineConnection::Open** method will be called for the line, setting up a TAPI opaque handle and returning the pointer to the **CTSPILineConnection** as our handle back to TAPI. For more information on this, see the section titled *Opaque handles*. A line will only be opened by TAPI once (regardless of how many applications request it). When the open call is received, the **CTSPIConnection::OpenDevice** method will be invoked. The reverse is true of the **Close** method. In this case, the **CTSPIConnection::CloseDevice** will be invoked.

In most third party implementations, the PBX connection is already open by the time the line device is opened by an application. The TSP++ library allows events to flow through the line objects regardless of their current open state. If TAPI doesn't have the line open then events will not flow through to TAPI.

Once the line is opened by TAPI, existing calls known to TSP++ (because of earlier events) will be automatically forwarded to TAPI.

Capabilities

The line capabilities of the line device are maintained in the **CTSPILineConnection** object. Generally, the capabilities are assigned during the initialization of the service provider. TAPI stores and expects the line device capabilities to be in a **LINEDEVCAPS** structure. The line object maintains a static copy of this structure for easy access.

A pointer to this structure can be obtained through the **GetLineDevCaps** method. If the SPLUI object storage system isn't used for auto-object creation, then the derived service provider will need to set required fields during the initialization process. These include:

Structure member	Description
dwDevCapFlags	Should be filled with applicable LINEDEVCAPFLAGS_
dwMaxNumActiveCalls	Should be adjusted if more than one call may be active on an address, or multiple addresses are shared on a channel.
dwBearerModes	If LINEBEARERMODE_PASSTHROUGH is supported, it needs to be or'ed into the bearer modes for the line. DO NOT include it with an address. The address bearer mode should always be a single mode.
dwAnswerMode	Should be filled if existing calls will not be dropped when answering a new call. This uses the LINEANSWERMODE_ bit flags
dwRingModes	Defaults to one.
dwUUIAcceptSize	Defaults to zero.
dwUUIAnswerSize	Defaults to zero.
dwUUIMakeCallSize	Defaults to zero.
dwUUIDropSize	Defaults to zero.

dwUUISendUserUserInfo Size	Defaults to zero.
dwUUICallInfoSize	Defaults to zero.
MinDialParams	Defaults to min/max parameters passed on CreateAddress.
MaxDialParams	Defaults to min/max parameters passed on CreateAddress.
DefaultDialParams	Must be supplied by derived class.

All the other fields are filled in either during initialization of the line object, or by the methods **CreateAddress** and **AddTerminal** which add capabilities to the device. If the field isn't listed, then it is set to a default value.

Device Specific Extensions

If the line device supports device-specific features, then these must be added either by overriding the **CServiceProvider** method **lineGetDevCaps**, or by overriding the **CTSPILineConnection** method **GatherCapabilities**. To add device-specific extensions to the **LINEDEVSTATUS** record, the **CServiceProvider::lineGetDevStatus**, or **CTSPILineConnection**::**GatherStatus** may be overridden.

Device function extensions can be supported by either overriding the **CServiceProvider** method **lineDevSpecific**, **lineDevSpecificFeature**, or the **CTSPILineConnection** method **DevSpecificFeature**.

> **Note:** Do not add the device specific information into the static LINEDEVCAPS, there is no extra-allocated space!

Status information

Status information for the line device is also contained within the **CTSPILineConnection** object. The TAPI server expects line status to be reported in a **LINEDEVSTATUS** structure. This structure is embedded in our line object to maintain the current state of the line. As different methods change the state of the line, TAPI is notified through the asynchronous callback of each changed element. Most notifications are supported completely by the library. The methods that change the status record are: **SetTerminalModes**, **SetRingMode**, **SetBatteryLevel**, **SetSignalLevel**, **SetRoamMode**, and **SetDeviceStatusFlags**. All fields except the device-specific information are automatically filled out by the line device and tracked through various callbacks by the address and call objects, and by using the above methods to change the information in the status record. The current line device status record may be obtained through the **GetLineDevStatus** method. Other caveats about status notifications are:

Ring Events

To support ring events in the line, the service provider class must call the **OnRingDetected** method. Each ring should call the method, and when the ringing stops, the method should be called one last time with a TRUE value set for the stop ringing flag. The ring count is managed automatically by the library.

Unsupported notifications

The notifications that are not directly supported by the library are: **LINEDEVSTATE_MAINTENANCE**, **LINEDEVSTATE_DEVSPECIFIC**, **LINEDEVSTATE_REINIT**, **LINEDEVSTATE_CONFIGCHANGE**, and **LINEDEVSTATE_TRANSLATECHANGE**. These can be supplemented by a derived service provider class and passed to TAPI by calling the **OnLineStatusChange** method.

Channels and Addresses

Each line has one or more *channels* on which incoming or outgoing calls may be placed. In a standard home telephone system (POTS), only one channel exists on a line, and it is used exclusively for voice or data traffic. With ISDN, at least three (up to 30+) channels can exist on a line simultaneously.

In the TSP++ library, each channel is assigned its own address. An address is corresponds to a telephone directory number. The library assigns each address a unique identifier called an Address ID to make it easy to identify. The identifiers range from zero to the total number of addresses present minus one. If two addresses share a channel with DID (direct inward dialing) or distinctive ring, two address object would still be added, but only one call could be active at any given time. Otherwise, the addresses work independently of each other and each address may have one or more active calls on the line.

The line object creates an address in response to the **CreateAddress** method. This method needs to be called during initialization of the service provider to add each address to the list of available addresses.

Terminals

The telephony hardware may have one or more *terminals* on which telephony signals are routed. This can include:

1. A hardware telephone device, possibly having lamps, buttons, display, etc. It must have some form of voice I/O (handset, speakerphone, etc.).
2. A connection to a PC sound card, or some other hardware routing mechanism for manipulating and interpreting the audio/telephony signals.

A TAPI application can control where the information is routed by using the **lineSetTerminal** function. This would impact the normal routing between the switch and the station, controlled by the TSP itself. This allows the TAPI application to turn specific audio devices on and off based on how the user wishes to interact with a call. For example, a piece of hardware might have a ringer and a headset associated with the line. The application can turn the ringer off and instead notify the user through a dialog that there is an incoming call. If the user decides to answer the call, the application can then cause all events to be routed to the headset, turning the speaker and microphone on and then asking the switch to route the audio signal to that terminal. The library supports the reporting and management of terminals through a series of methods, **AddTerminal**, **RemoveTerminal**, **GetTerminalCount**, and **GetTerminalInformation**. TAPI is automatically informed about all terminal devices that are added using these APIs. When the command to route terminal information is invoked by an application using **lineSetTerminal**, a **RTSetTerminal** request is placed into the queue.

When the service provider processes the request, the **SetTerminalModes** method should be used to inform the class library about the new assignments for each terminal.

This will cause the class library to send all the appropriate notifications and adjust each of the active calls on the line to reflect the proper terminal information.

Request completions

When a request completes on the line, the **OnRequestComplete** method is called. The default line connection object takes the following actions when processing requests:

RTSetTerminal If the request completes successfully, the line connection will alter the internal terminal map and notify TAPI that the terminal information has changed in the **LINEDEVCAPS** record. This change will be cascaded to the address and calls that are owned by the line.

RTForward If the request completes successfully, forwarding record(s) will be added to the **LINEDEVCAPS** structure and TAPI will be notified that the information has changed. All addresses on the line will be adjusted as well.

REQEST_MEDIACONTROL If the request completes successfully, the given media structures are stored into the line and all associated address objects for later retrieval by TAPI.

RTCompleteCall If the request completes successfully, a call completion request will be added to an internal array for notification of the final completion. It may be located using the **FindCallCompletionRequest** and deleted using the **RemoveCallCompletionRequest** methods.

RTUncompleteCall If the request completes successfully, the call completion request that was queued originally will be removed and deleted. TAPI is not notified since this is not considered a *device* request, rather a user request.

Constructor and Destructor

CTSPILineConnection	Constructor for the line connection object.
~CTSPILineConnection	Destructor for the line connection object.

Operations - Public Members

AddTerminal	Add a new terminal device to the line (*initialization only!*)
CreateAddress	Create a new address on the line (*initialization only!)*
CreateMonitoredAddress	Creates a new monitored address on the line (*initialization only!).* This is where the address is shared (**LINEADDRESSSHARING_MONITORED**).
CreateUIDialog	Creates a new asynchronous user-interface dialog associated with an open line.
DeleteCallsOnClose	Turns on a flag to automatically delete all call objects when the line object is closed by TAPI. This is to facilitate switches which have some ability to scan for existing calls when a line is opened and can

	significantly improve performance when a large number of lines are supported.
DevLocked	Turn the **LOCKED** bit on and off in the device status flags
DevMsgWaiting	Turn the **MSGWAITING** bit on and off in the device status flags
DevStatusConnected	Turn the **CONNECTED** bit on and off in the device status flags
DevStatusInService	Turn the **INSERVICE** bit on and off in the device status flags.
EnableAgentProxy	Enable the **JTAPROXY** for this line. This allows agent commands to be received for this particular line device.
FindAddress	Locates a child **CTSPIAddressInfo** object from the dialable number associated to it.
FindCallCompletionRequest	Locate a call completion request by call, switch information, or call completion ID.
FindCallByID	Locates a call appearance on this line based on the call-id.
FindCallByState	Locates a call appearance on this line based on a call state.
FindCallByType	Locates a call appearance on this line based on the type of call (consultation, conference, regular).
ForceClose	Forces the line to close immediately
GetAddress	Locate an address object by its dialable address or id.
GetAddressCount	Returns the number of addresses on this line.
GetAssociatedPhone	Returns the **CTSPIPhoneConnection** associated with this line device.
GetDefaultMediaDetection	Returns the default media types detected
GetLineHandle	Returns the TAPI opaque line handle for this device.
GetLineDevCaps	Returns a pointer to the **LINEDEVCAPS** for this line.
GetLineDevStatus	Returns a pointer to the **LINEDEVSTATUS** for this line
GetLineType	Returns the line type for this line device.
GetTerminalCount	Returns the count of terminals on this line.
GetTerminalInformation	Returns the information associated with a terminal on this line.
GetUIDialogItem	Returns the 32-bit item-data stored with a invoked user-interface item.
OnRingDetected	This method needs to be called whenever a ring is detected on this line.
OnUIDialogClosed	This method is called when a user-interface dialog is closed by the interactive session.

RemoveCallCompletionRequest	Remove and delete a call completion request.
RemoveTerminal	Remove an existing terminal from the line.
SendDialogInstanceData	Sends data to an open user-interface object.
Send_TAPI_Event	This method sends TAPI an event record through the asynchronous callback supplied by TAPI on the **Open**.call.
SetBatteryLevel	This method sets the current battery level in the **LINEDEVSTATUS** record.
SetDeviceStatusFlags	This method sets the current device flags in the **LINEDEVSTATUS** record.
SetRingMode	This method sets the current ring mode listed in the **LINEDEVSTATUS** record.
SetRoamMode	This method sets the current roaming mode in the **LINEDEVSTATUS** record.
SetLineFeatures	This allows the service provider to adjust the line features in the **LINEDEVSTATUS** record.
SetMediaControl	This method is called when a **lineSetMediaControl** request completes successfully or the media event management is changing on the line.
SetMSPGUID	This sets the unique GUID to represent the media service provider for this TSP.
SetPermanentLineID	Sets the permanent device identifier associated with this line device.
SetSignalLevel	This method sets the current signal level in the **LINEDEVSTATUS** record.
SetTerminalModes	This method sets the terminal routing information. This is called by the library when a **RTSetTerminal** completes successfully.
SupportsAgents	Returns whether the **JTAPROXY** program is running for this line device.

Operations - Protected Members

SetDeviceID	This is called when TAPI assigns the final device id for the line. It is used for dynamic line creation and generally should not be used outside TSP++.

Overridables - Public Members

CanSupportCall	Return whether the call-type can be supported on this line. This is used internally to validate **LINECALLPARAMS**.
FindAvailableAddress	Locate an address suitable for carrying a call type.
GetPermanentDeviceID	Returns the permanent device ID assigned by the class library to this line device. This device ID is a

	combination of the provider ID and the position within the device array of the **CTSPIDevice** object owner.
OnMediaConfigChanged	This is called when the media configuration information is changed.
OnLineCapabilitiesChanged	This is called when any data in the **LINEDEVCAPS** record changes.
OnLineStatusChange	This is called when any data in our **LINEDEVSTATUS** record changes to notify TAPI.
read	Function to read additional information from a registry object. Used when the SPLUI object-storage system is utilized for configuration.
ValidateMediaControlList	Validate the media control list against our **LINEDEVCAPS**. This is used internally during **lineSetMediaControl**.

Operations - Protected Members

FreeDialogInstance	Frees an open user-interface object.
IsConferenceAvailable	Returns a bool indicating whether there is an active conference call on this line.
IsTransferConsultAvailable	Returns a bool indicating whether there is a consultation transfer pending on this line.

Overridables - Protected Members

ConvertDialableToCanonical	This method is used to convert a dialable number into its canonical format. It may be overridden to support various international requirements.
GetRequestMap	This internal function is used to locate the request map which routes TAPI events. It may be overridden to change the association of request maps on a per-line basis.
Init	This method is called directly after the constructor by the **CTSPIDevice** owner.
OnAddressFeaturesChanged	This is called when address features are changed within an owned address object on this line.
OnCallDeleted	Called when a **CTSPICallAppearance** object is being deleted (deallocated) by an address. The default implementation removes any references to the call from any pending or working asynchronous requests.
OnCallFeaturesChanged	This is called by the address object when a call on the address changes its call features.
OnCallStateChange	This is called by call appearances on this line when

	the call state information changes after TAPI has been notified.
OnConnectedCallCountChange	This is called when the connected call count changes in any of the lines address objects.
OnLineFeaturesChanged	This is called when the line features are changed by the class library.
OnMediaControl	This is called when the media control information is changed for the line device.
OnPreCallStateChange	This is called by call appearances on this line when their call state information is changing. This is called before TAPI is notified.
OnRequestComplete	This method is called as a request on this line completes. It performs various cleanup procedures required for the request based on whether the request finished successfully or not.
RecalcLineFeatures	This is called to recalculate the line features for the device.

Overridables - TAPI Members

Close	This method is called for **lineClose**. Default implementation decrements the device count and waits for all **Drop** requests to finish.
ConditionalMediaDetection	This method is called for **lineConditionalMediaDetection**. Handled completely within the library.
DevSpecific	This method is called for **lineDevSpecific**. Default implementation returns **Not Supported**.
DevSpecificFeature	This method is called for **lineDevSpecificFeature**. Default implementation returns **Not Supported**.
Forward	This method is called for **lineForward**. Default implementation issues a **RTForward** request.
GatherCapabilities	This method is called for **lineGetDevCaps**. Handled completely within the library.
GatherStatus	This method is called for **lineGetDevStatus**. Handled completely within the library.
GenericDialogData	This is called to process **lineGenericDialogData** requests on the line.
GetAddressID	This method is called for **lineGetAddressID**. Managed completely within the library.
GetCallHubTracking	This method is called to return the current state of the call-hub tracking for TAPI 3.0.
GetDevConfig	This method is called for **lineGetDevConfig**.. Default implementation returns **Not Supported**.
MakeCall	This method is called for **lineMakeCall**. Default

	implementation locates an available call appearance and issues a **RTMakeCall** request.
MSPIdentify	This method is called to identify the media service provider associated with this TSP for TAPI 3.0.
Open	This method is called for **lineOpen**. Default implementation increments the device count and swaps handles with TAPI.
ReceiveMSPData	This method is called when the associated media service provider sends information to the TSP under TAPI 3.0.
SetCallHubTracking	This method implements the call-hub tracking support for TAPI 3.0.
SetDefaultMediaDetection	This method is called for **lineSetDefaultMediaDetection**. Handled completely within the library.
SetDevConfig	This method is called for **lineSetDevConfig**.. Default implementation returns **Not Supported**.
SetLineDevStatus	This method is called in response to a **lineSetLineDevStatus** function.
SetStatusMessages	This method is called for **lineSetStatusMessages**. Handled completely within the library.
SetTerminal	This method is called for **lineSetTerminal**. Default implementation issues a **RTSetTerminal** request.
SetupConference	This is called for **lineSetupConference**. Default implementation issues a **RTSetupConference** request.
SetupTransfer	This is called for **lineSetupTransfer**. Default implementation issues a **RTSetupTransfer** request.
UncompleteCall	Remove a call completion request for this line.

CTSPILineConnection::AddTerminal

int AddTerminal(LPCTSTR *lpszName*, **LINETERMCAPS&** *termCaps*,
DWORD *dwModes* **= 0L);**

lpszName	Text name of the terminal device
termCaps	Terminal capabilities to associate with the terminal
dwModes	**LINETERMMODE_xxx** flags to associate with terminal.

Remarks

This method creates a new terminal device for the line. A terminal device is an end-point where all line-oriented telephony signals are routed. This can include buttons, tones, audio signals, display information, etc.

This method automatically adds the structures necessary to report the terminal information back to TAPI, and if **lineSetTerminal** is invoked on the line, the information passed will be checked against the available added terminals.

TAPI is notified that the number of terminals has changed through the **LINEDEVSTATE_TERMINALS** event.

Return Value

Terminal identifier – this is simply the position of the entry within the terminal array. If the method fails, (-1) is returned.

CTSPILineConnection::CanSupportCall

virtual LONG CanSupportCall (
const LPLINECALLPARAMS *lpCallParams***) const;**

lpCallParams **LINECALLPARAMS** structure to verify.

Remarks

This method is called to verify that a call of a specified type may be supported by the line object. The method by default runs through all the **CTSPIAddressInfo** objects and calls the **CanSupportCall** method of each address.

Return Value

This method returns a standard TAPI return code. If any of the address objects report that the call can be made, then the method returns successful.

If none of the addresses can handle the call based on the information in the structure, then the method returns either a specific error as to what cannot be supported, or **LINEERR_INVALCALLPARAMS**.

CTSPILineConnection::Close

virtual LONG Close();

Remarks

This method is invoked by **TAPI** when the line is closed by all applications. It calls the **CTSPIDevice::CloseDevice** method and resets all the line handle information. Once this method completes, the line will have no further interaction with TAPI until it is opened again.

Return Value

This method returns a standard TAPI return code.

CTSPILineConnection::ConditionalMediaDetection

virtual LONG ConditionalMediaDetection(DWORD *dwMediaModes***,**
const LPLINECALLPARAMS *lpCallParams***);**

dwMediaModes Media modes which are being checked by **LINEMAPPER**.
lpCallParams **LINECALLPARAMS** structure to verify.

Remarks

This method is invoked by **TAPI** when the application requests a line open using the **LINEMAPPER**. This method will check the requested media modes and return an acknowledgement based on whether the line can monitor all the requested modes and can support the request call type based on the **LINECALLPARAMS** structure.

Return Value

This method returns a standard TAPI return code. It calls the **CServiceProvider::ProcessCallParameters** method to validate the call parameters. The media mode item within the call parameters will be the media modes passed into this method.

If the call parameters are supported, zero is returned.

CTSPILineConnection::ConvertDialableToCanonical

virtual TString ConvertDialableToCanonical(LPCTSTR *pszPartyID*,
DWORD *dwCountryCode*, **bool** *fInbound*=**false);**

pszPartyID	Dialable number.
dwCuntryCode	Country code for the number
fInbound	TRUE if this is an inbound call

Remarks

This method is used to convert a dialable number to a canonical number. It can be overridden to provide line-specific conversions. The default implementation forwards the request to the **CServiceProvider** implementation of this function.

Return Value

This method returns the canonical representation of the given dialable number. For more information on dial strings, see the section on *Dialing Information* or the section on *Structures: DIALINFO*.

CTSPILineConnection::CreateAddress

CTSPIAddressInfo* CreateAddress (LPCTSTR *lpszDialableAddr* **= NULL,**
LPCTSTR *lpszAddrName* **= NULL,**
bool *fAllowIncoming* **= TRUE,**
bool *fAllowOutgoing* **= true,**
DWORD *dwAvailMediaModes* **= LINEMEDIAMODE_UNKNOWN,**
DWORD *dwBearerMode* **= LINEBEARERMODE_VOICE,**
DWORD *dwMinRate* **= 0L, DWORD** *dwMaxRate* **= 0L,**
LPLINEDIALPARAMS *lpDialParams* **= NULL,**
DWORD *dwMaxNumActiveCalls* **= 1,**
DWORD *dwMaxNumOnHoldCalls* **= 0,**
DWORD *dwMaxNumOnHoldPendCalls* **= 0,**
DWORD *dwMaxNumConference* **= 0,**
DWORD *dwMaxNumTransConf* **= 0**
DWORD *dwAddressType* = **0);**

lpszDialableAddr	Dialable phone number of the address.
lpszAddrName	Textual name reported back in **LINEADDRESSCAPS** structure.

fAllowIncoming	**TRUE** if incoming calls are allowed.
fAllowOutgoing	**TRUE** if outgoing calls are allowed.
dwAvailMediaModes	Available media modes on this address.
dwBearerMode	Single **LINEBEARERMODE_xxx** flag.
dwMinRate	Minimum data rate reported in **LINEADDRESSCAPS**.
dwMaxRate	Maximum data rate reported in **LINEADDRESSCAPS**.
lpDialParams	Dialing parameters (**NULL** to use line information).
d*wMaxNumActiveCalls*	Max number of calls in a **Connected** state.
d*wMaxNumOnHoldCalls*	Max number of calls in a **Hold** state.
dwMaxNumOnHoldPendCalls	Max number of calls waiting for **Transfer**/**Conference**.
d*wMaxNumConference*	Max number of calls conferenced together.
d*wMaxNumTransConf*	Max number of calls conferenced from a transfer event.
dwAddressType	The **LINEADDRESSTYPE_xxx** for this address (TAPI 3.0 only).

Remarks

This method is used to create a new address on a line. It should be used during the initialization of the line object (**CTSPILineConnection::Init**). The information given to each address is used to determine the capabilities of the line itself. For instance, all the media modes for each added address object are collected and returned in the **LINEDEVCAPS** of the line object.

Return Value

A pointer to the created address object, NULL if there was an error.

CTSPILineConnection::CreateMonitoredAddress

```
CTSPIAddressInfo* CreateMonitoredAddress (
        LPCTSTR lpszDialableAddr = NULL,
        LPCTSTR lpszAddrName = NULL,
        DWORD dwAvailMediaModes = LINEMEDIAMODE_UNKNOWN,
        DWORD dwBearerMode = LINEBEARERMODE_VOICE,
        DWORD dwMinRate = 0L, DWORD dwMaxRate = 0L,
        DWORD dwMaxNumActiveCalls = 1,
        DWORD dwMaxNumOnHoldCalls = 0,
        DWORD dwMaxNumOnHoldPendCalls = 0,
        DWORD dwAddressType = 0);
```

lpszDialableAddr	Dialable phone number of the address.
lpszAddrName	Textual name reported back in **LINEADDRESSCAPS** structure.
dwAvailMediaModes	Available media modes on this address.
dwBearerMode	Single **LINEBEARERMODE_xxx** flag.
dwMinRate	Minimum data rate reported in **LINEADDRESSCAPS**.
dwMaxRate	Maximum data rate reported in **LINEADDRESSCAPS**.

d*wMaxNumActiveCalls*	Max number of calls in a **Connected** state.
d*wMaxNumOnHoldCalls*	Max number of calls in a **Hold** state.
dwMaxNumOnHoldPendCalls	Max number of calls waiting for **Transfer**/**Conference**.
dwAddressType	The **LINEADDRESSTYPE_xxx** for this address (TAPI 3.0 only).

Remarks

This method is used to create a new monitored address on a line. A monitored address is one that is shared or bridged on the hardware.

This method should be used during the initialization of the line object (**CTSPILineConnection::Init**). The information given to each address is used to determine the capabilities of the line itself. For instance, all the media modes for each added address object are collected and returned in the **LINEDEVCAPS** of the line object.

Return Value

A pointer to the created address object, NULL if there was an error.

CTSPILineConnection::CreateUIDialog

HTAPIDIALOGINSTANCE CreateUIDialog (DRV_REQUESTID *dwRequestID***,**
LPVOID *lpParams* **= NULL, DWORD** *dwSize* **= 0L,**
LPCTSTR *lpszUIDLLName* **= NULL);**

dwRequestID	Request ID which identifies the process owner of the dialog.
lpParams	Optional pointer to a parameter block
dwSize	Size of the above parameter block
lpszUIDLLName	Name of the user-interface DLL to load (**NULL** for default).

Remarks

This method causes TAPI to load a unsolicited dialog in the process space of the application that sponsored the **dwRequestID** parameter. The request ID must be valid at the time of this API being called (i.e. the request cannot have been completed yet). The parameter block is user-defined, so any information may be passed to the dialog. If no DLL name is supplied, then the name passed to the constructor of the **CServiceProvider** object is used.

Return Value

Handle to the dialog created or NULL if there was an error.

CTSPILineConnection::CTSPILineConnection

CTSPILineConnection()

Remarks

This is the constructor for the line connection object.

CTSPILineConnection::~CTSPILineConnection

~CTSPILineConnection()

Remarks

This is the destructor for the line connection object.

CTSPILineConnection::DeleteCallsOnClose

void DeleteCallsOnClose();

Remarks

This method sets a flag on the line object to indicate that it doesn't need to track call progress when the line is not opened by TAPI. The default behavior is to continue to track calls and progression messages from the hardware so that the information is available when/if TAPI opens the line.

If the hardware in question has a query facility to determine all the active calls on a line, then there is no need for TSP++ to track calls when they are not used by TAPI. In this case, this method can be invoked during line initialization and all the call objects will be deleted when the line is closed. TSP++ will then expect the derived TSP code to re-create all the active calls (through the **CreateCallAppearance** method) when the line is re-opened (during the **Open** method).

Important Note: This function does not drop the calls, it simply deletes the internal object representation of the calls from memory. This will happen during the **Close** method if this function was called during the line initialization process.

CTSPILineConnection::DevLocked

void DevLocked(bool *fLocked***=true);**

fLocked Whether to set/clear the locked bit.

Remarks

This method is used to set or clear the **LINEDEVSTATUS_LOCKED** bit in the line device status flags.

CTSPILineConnection::DevMsgWaiting

void DevMsgWaiting(bool *fMsgWaiting***=true);**

fMsgWaiting Whether to set/clear the message waiting bit.

Remarks

This method is used to set or clear the **LINEDEVSTATUS_MSGWAITING** bit in the line device status flags.

CTSPILineConnection::DevSpecific

virtual LONG DevSpecific(CTSPIAddressInfo* *pAddr*,
CTSPICallAppearance* *pCall*, **DRV_REQUESTID** *dwRequestID*,
LPVOID *lpParam*, **DWORD** *dwSize*);

pAddr	Address this request is for (may be NULL).
pCall	Call this request is for (may be NULL).
dwRequestID	Asynchronous request ID associated with this request.
lpParams	Optional pointer to a parameter block
dwSize	Size of the above parameter block

Remarks

This method is called by the library when an application calls **lineDevSpecific** and identifies this line object in the **HLINE** parameter. The **TSPI_lineDevSpecific** function must be exported from the provider. This method enables service providers to provide access to features not typically offered by TAPI. The meaning of these extensions are device specific and known only to the service provider and applications which are written to take advantage of them. This method is not handled directly by the library and is provided simply for overriding purposes.

Return Value

You must override this method to provide device-specific functionality. The library returns **LINEERR_OPERATIONUNAVAIL** if you do not override this method.

CTSPILineConnection::DevSpecificFeature

virtual LONG DevSpecificFeature(DWORD *dwFeature*,
DRV_REQUESTID *dwRequestID*,
LPVOID *lpParams*, **DWORD** *dwSize*);

dwFeature	Feature code passed to **lineDevSpecificFeature**.
dwRequestID	Asynchronous request ID associated with this request.
lpParams	Optional pointer to a parameter block
dwSize	Size of the above parameter block

Remarks

This method is called by the library when an application calls **lineDevSpecificFeature** and identifies this line object in the **HLINE** parameter. The **TSPI_lineDevSpecificFeature** function must be exported from the provider. This method enables service providers to provide access to features not typically offered by TAPI. The meaning of these extensions are device specific and known only to the service provider and applications which are written to take advantage of them. This method is not handled directly by the library and is provided simply for overriding purposes.

Return Value

You must override this method to provide device-specific functionality. The library returns **LINEERR_OPERATIONUNAVAIL** if you do not override this method.

CTSPILineConnection::DevStatusConnected

void DevStatusConnected(bool *fConnected*=**true);**

fConnected Whether to set/clear the connected bit.

Remarks

This method is used to set or clear the **LINEDEVSTATUS_CONNECTED** bit in the line device status flags.

CTSPILineConnection::DevStatusInService

void DevStatusInService(bool *fInservice*=**true);**

fInService Whether to set/clear the in-service bit.

Remarks

This method is used to set or clear the **LINEDEVSTATUS_INSERVICE** bit in the line device status flags.

CTSPILineConnection::EnableAgentProxy

void EnableAgentProxy();

Remarks

This method activates agent features for this line using the supplied **JTAPROXY** application.

JTAPROXY does not need to be running when this is called – if it is not running, then when it becomes available the line will be registered and ready for agent activity.

Note that until **JTAPROXY** is running, agent features are disabled.

CTSPILineConnection::FindAddress

CTSPIAddressInfo* FindAddress(LPCTSTR *lpszDialableAddr*) **const;**

lpszDialableAddr Dialable number to match against extensions.

Remarks

This method searches the address list associated with this line and looks for an address whose extension (**GetDialableAddress**) matches the passed dialable number.

Return Value

The **CTSPIAddressInfo** object identified by the dialable number or NULL if no address matches the input.

CTSPILineConnection::FindAvailableAddress

virtual CTSPIAddressInfo* FindAvailableAddress (
const LPLINECALLPARAMS *lpCallParams*,
DWORD *dwFeature* **= 0) const;**

lpCallParams	**LINECALLPARAMS** structure which identifies the type of call.
dwFeature	Feature which will use the address (**LINEADDRFEATURE_xxx**)

Remarks

This method is used in the library to locate a specific address to use for an outgoing call. Either the address identified by the **LINECALLPARAMS.dwAddressID** is returned, or the information within the **LINECALLPARAMS** structure is used to find the best suitable address by bearer mode and media mode information.

Return Value

A **CTSPIAddressInfo** object that can handle the call type presented within the **LINECALLPARAMS** structure.

CTSPILineConnection::FindCallByID

CTSPICallAppearance* FindCallByCallID(DWORD *dwCallID***) const;**

dwCallID	Call-id to search for.

Remarks

This method is used in the library to locate a call on the given line with the specified call-id.

Return Value

The **CTSPICallAppearance** found which has the specified call-id. NULL if no call is found.

CTSPILineConnection::FindCallByState

CTSPICallAppearance* FindCallByState(DWORD *dwState***);**

dwState	Call state to locate.

Remarks

This method is used in the library to locate any call on the line that is in the given state. The first call found is returned. Each address on the line is checked in order to find a call.

Return Value

The first **CTSPICallAppearance** found which is in the given state.

CTSPILineConnection::FindCallByType

CTSPICallAppearance* FindCallByType(int iType) const;

iType	Call type to search for **CTSPICallAppearance::**(Normal, Conference, Consultant).

Remarks

This method is used in the library to locate the first call on the line that is of the specified type. Each address is checked in order.

Return Value

The first **CTSPICallAppearance** found which is of the specified type.

CTSPILineConnection::FindCallCompletionRequest

RTCompleteCall* FindCallCompletionRequest (DWORD *dwSwitchInfo***,**
LPCTSTR *pszSwitchInfo***);**

RTCompleteCall* FindCallCompletionRequest(
CTSPICallAppearance* *pCall***);**

RTCompleteCall* FindCallCompletionRequest(
DWORD *dwCompletionID***);**

dwSwitchInfo	Switch information which was associated with a completion request.
pszSwitchInfo	ASCII Switch information which was associated with a completion request.
pCall	Call object which is associated with completion request.
dwCompletionID	Completion identifier associated with completion request.

Remarks

These methods are provided to locate call completion requests that have been setup on the line. Call completion requests specify how a call that could not be connected normally should be completed instead. They are issued by the application using **lineCompleteCall**. The network or switch might be unable to complete a call because the destination is busy, doesn't answer, or because there are no outgoing lines available on the PBX. The completion request is created by the **CompleteCall** asynchronous request and stays in an array owned by the line object.

This method allows the provider to locate a completion request once the call has come back to the station – when the PBX is signaling the final completion of the call. A new call will typically be created and offered to the application using **CTSPIAddressInfo::CreateCallAppearance** with the reason code being **LINECALLREASON_CALLCOMPLETION**, and the completion id filled out from the **RTCompleteCall** record.

The completion record should be destroyed using the **RemoveCallCompletionRequest** by the service provider when the completion is detected and offered on the PBX.

Return Value

The **RTCompleteCall** record which matches the given search criteria.

CTSPILineConnection::ForceClose

void ForceClose();

Remarks

This method may be called by the service provider to close the line device immediately regardless of it being in-use or not. It should be used when the device is going offline for some hardware reason or because a component or network connection has been lost.

CTSPILineConnection::Forward

virtual LONG Forward(DRV_REQUESTID *dwRequestId***,**
CTSPIAddressInfo* *pAddr***, TForwardInfoArray*** *parrForwardInfo***,**
DWORD *dwNumRingsNoAnswer***, HTAPICALL** *htConsultCall***,**
LPHDRVCALL *lphdConsultCall***, LPLINECALLPARAMS** *lpCallParams***);**

dwRequestID	Asynchronous request ID associated with this forward request.
pAddr	Address object to forward (NULL means all addresses)
lpForwardInfoArray	Array of **TSPIFORWARDINFO** structures (see the section on *Data Structures*).
dwNumRingsNoAnswer	The number of rings to wait before forwarding a call from this address.
htConsultCall	TAPI opaque call handle for the creation of a consultation call as a result of this operation.
lphdConsultCall	TSP call handle to match to the created consultation call created for the forwarding.
lpCallParams	TAPI Call parameters to use for this newly created consultation call.

Remarks

This method is called when the line is forwarded using **lineForward**. The **TSPI_lineForward** function must be exported from the provider. The **TSPILINEFORWARD** structures contain all the parameters that were sent by the application. The consultation call parameters are used only if the network or switch creates a consultation call because of a forwarding request. In this case, the service provider is responsible for creating a new call appearance using **CTSPIAddressInfo::CreateCallAppearance** passing in the **htConsultCall**, and assigning the resultant pointer to the **lphdConsultCall** pointer.

The default behavior of the method is to either pass the request to the given address object, or if that is NULL, verify that all the addresses can be forwarded using the **CTSPIAddressInfo::CanForward** method, and then add an asynchronous request of type **RTForward** into the queue.

Return Value

TAPI Error code or asynchronous request ID if the request was queued.

CTSPILineConnection::FreeDialogInstance

LONG FreeDialogInstance(HTAPIDIALOGINSTANCE *htDlgInst***);**

htDlgInst	Dialog instance to deallocate.

Remarks

This method is called when a unsolicited user-interface dialog which was created using **CreateUIDialog** is being deallocated due to it being destroyed. This method removes all references to the dialog and deletes the internal representation of the dialog from the provider.

Return Value

TAPI Error code or zero if the method was successful.

CTSPILineConnection::GatherCapabilities

virtual LONG GatherCapabilities (DWORD *dwTSPIVersion***,**
DWORD *dwExtVer***, LPLINEDEVCAPS** *lpLineCaps***);**

dwTSPIVersion	The TAPI version which structures should reflect.
dwExtVer	The extension version which structures should reflect.
lpLineCaps	The structure to fill with line capabilities.

Remarks

This method is called when an application requests line capabilities using the **lineGetDevCaps** function. The class library automatically fills in all the known information about the provider using the line, address, and call objects which are associated with the current line object.

The service provider can adjust the capabilities returned by using the **GetLineDevCaps** method and modifying the returning structure. This should typically be done when the line is first initialized as capabilities normally do not change during the life of the provider.

If the structure is modified through the **GetLineDevCaps** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **LINEDEVCAPS** structure.

Return Value

TAPI Error code or zero if the method was successful.

CTSPILineConnection::GatherStatus

virtual LONG GatherStatus (LPLINEDEVSTATUS *lpStatus***);**

lpStatus	The structure to fill with line status.

Remarks

This method is called when an application requests the current line status using the **lineGetDevStatus** function. The class library automatically fills in all the known information about the provider using the line, address, and call objects which are associated with the current line object.

The service provider can adjust the capabilities returned by using the **GetLineDevStatus** method and modifying the returning structure. Some of the information can be modified using other **CTSPILineConnection** methods. These include **SetBatteryLevel**, **SetSignalLevel**, **SetRoamMode**, **SetDeviceStatusFlags**, **SetRingMode**, and **SetTerminalModes**. Using these methods is recommended as TAPI is notified about the information changing through a **LINEDEVSTATUS** callback.

If the structure is modified through the **GetLineDevStatus** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **LINEDEVSTATUS** structure.

Return Value

TAPI error code or zero if the method was successful.

CTSPILineConnection::GenericDialogData

virtual LONG CTSPILineConnection::GenericDialogData (
LPVOID *lpvItemData***, LPVOID** *lpParam***, DWORD** *dwSize***);**

lpvItemData	32-bit item-data passed to the **CreateUIDialog** method.
lpParam	Parameter from the UI dialog.
dwSize	Size of the passed parameter block.

Remarks

This method is called when the user-interface component of the TSP is sending data back to the service provider, and the object type specified in the **TSPI_providerGenericDialogData** function was set to **TUISPIDLL_OBJECT_LINEID**.

Return Value

Standard TAPI return code. FALSE indicates success.

CTSPILineConnection::GetAddress

CTSPIAddressInfo* GetAddress (DWORD *dwAddressID***) const;**

dwAddressID	Numeric address id (zero-based) to locate.

Remarks

This method searches the address array associated with the line connection and locates the address associated with the given parameter.

Return Value

Address object associated with the given parameter, NULL if not found.

CTSPILineConnection::GetAddressCount

DWORD GetAddressCount() const;

Remarks

This method returns the number of addresses that were created on the line using the **CreateAddress** method.

Return Value

Number of addresses on the line, this number will be one larger than the largest index that may be passed into the **GetAddress** method.

CTSPILineConnection::GetAddressID

virtual LONG GetAddressID(LPDWORD *lpdwAddressId***,**
DWORD *dwAddressMode***, LPCTSTR** *lpszAddress***, DWORD** *dwSize***);**

lpdwAddressID	Returning address id which was found.
dwAddressMode	Type of address specified in **lpszAddress**.
lpszAddress	Dialable phone number of the address to locate.
dwSize	Size of the **lpszAddress** string.

Remarks

This method returns the address identifier associated with an address in the specified format on the line. This method is called in response to an application calling **lineGetAddressID**.

The default behavior for this method is to enumerate through all the addresses on the line object and locate the one that has the specified dialable address.

Return Value

TAPI error code or zero if the method was successful.

CTSPILineConnection::GetAssociatedPhone

CTSPIPhoneConnection* GetAssociatedPhone() const;

Remarks

This method returns the phone object that has been associated to this line device. This association is performed using the **CTSPILineDevice:: AssociateLineWithPhone** method.

Return Value

Phone object pointer or NULL if this line does not have a phone assigned to it.

CTSPILineConnection::GetCallHubTracking

virtual LONG GetCallHubTracking(
LPLINECALLHUBTRACKINGINFO *lpTrackingInfo***);**

lpTrackingInfo Returning call hub data structure.

Remarks

This method is called by TAPI 3.0 to get the current call hub tracking data for the line. The **TSPI_lineGetCallHubTracking** function must be exported for this to be called.

Return Value

TAPI error code or zero if the function was successful.

CTSPILineConnection::GetDefaultMediaDetection

DWORD GetDefaultMediaDetection() const;

Remarks

This method should be used by the service provider to determine that media modes that TAPI is interested in being notified about. Offering calls should not be presented to TAPI unless the media mode is present in the return value of this method.

When writing for third party telephony under TAPI 2.x, all calls should be offered to TAPI, and this method should only be used to see which ones have active owners.

Return Value

Last known media mode that was set by TAPI through the **TSPI_lineSetDefaultMediaDetection**.

CTSPILineConnection::GetDevConfig

virtual LONG GetDevConfig(TString& *strDeviceClass*,
LPVARSTRING *lpDeviceConfig*);

strDeviceClass Device class which is being asked for.
lpDeviceConfig Returning device configuration information.

Remarks

This method is called in response to an application calling **lineGetDevConfig**. It is used for device-specific extensions that are not provided through the general TAPI interface. It is not directly supported in the class library.

The service provider must override this method in order to provide the device-specific capability. The default implementation of the method returns an error.

Return Value

The class library returns **LINEERR_OPERATIONUNAVAIL**.

CTSPILineConnection::GetLineDevCaps

LPLINEDEVCAPS GetLineDevCaps();

Remarks

This method returns the **LINEDEVCAPS** structure that is maintained for the line and returned by the **GatherCapabilities** method. Any of the non-pointer information may be read or modified through the resulting pointer, although TAPI is not notified if the information is modified directly. To make sure TAPI is correctly updated, use the appropriate member method of the **CTSPILineConnection** object to modify the capabilities of the line.

WARNING: Do not attempt to add information to the end of the structure, use the provided **CTSPILineConnection** methods to associate the additional pointer information with the line.

Return Value

Pointer to the **LINEDEVCAPS** structure.

CTSPILineConnection::GetLineDevStatus

LPLINEDEVSTATUS GetLineDevStatus();

Remarks

This method returns the **LINEDEVSTATUS** structure that is maintained for the line and returned by the **GatherStatus** method. Any of the non-pointer information may be read or modified through the resulting pointer, although TAPI is not notified if the information is modified directly. To make sure TAPI is correctly updated, use the appropriate member method of the **CTSPILineConnection** object to modify the status of the line.

If the service provider *does* modify the structure, and wants to notify TAPI, the **CTSPILineConnection::OnLineStatusChange** method should be used.

WARNING: Do not attempt to add information to the end of the structure, use the provided **CTSPILineConnection** methods to associate the additional pointer information with the line.

Return Value

Pointer to the **LINEDEVSTATUS** structure.

CTSPILineConnection::GetLineHandle

HTAPILINE GetLineHandle() const;

Remarks

This method returns the line handle that was assigned to this line device by TAPI when the line was opened.

Return Value

Line handle assigned by TAPI, or zero if the line is not currently open.

CTSPILineConnection::GetLineType

int GetLineType() const;

Remarks

This method returns the line type for this line device. This will be one of the following:

Station	*Queue*	*RoutePoint*	*PredictiveDialer*
VRU	*Trunk*	*Other*	

Return Value

The line type (enumerated constant of **CTSPILineConnection**).

CTSPILineConnection::GetRequestMap

Protected

virtual TRequestMap* GetRequestMap() const;

Remarks

This method is called by the library to locate the request map which is used to route TAPI events throught the service provider (see **BEGIN_TSPI_REQUEST** and **END_TSPI_REQUEST**). The default behavior returns the class-global request map.

Return Value

Pointer to the internal request map for this object.

CTSPILineConnection::GetTerminalCount

int GetTerminalCount() const;

Remarks

This method returns the number of terminals that were added to this line using the **AddTerminal** method. For more information on terminals, see the section on *Terminals* in the line object description above.

Return Value

Count of terminals that are associated with this line object (zero if none).

CTSPILineConnection::GetTerminalInformation

DWORD GetTerminalInformation (int *iTerminalID***) const;**

iTerminalID Terminal to return information for.

Remarks

This method returns the **LINETERMMODE_xxx** constants that are associated with the specified terminal identifier. The passed identifier should be between zero and the total count of terminals on the line.

Return Value

The **LINETERMMODE_xxx** constants that were associated with the terminal when it was added to the line.

CTSPILineConnection::GetUIDialogItem

LPVOID GetUIDialogItem(HTAPIDIALOGINSTANCE *htDlgInst***) const;**

htDlgInstance	TAPI dialog instance to locate

Remarks

This method locates and returns the 32-bit item-data value that is associated with the specified unsolicited user-interface dialog specified. The dialog must have been created using the **CreateUIDialog** method.

Return Value

The 32-bit value associated with the given TAPI handle, or NULL if no dialog was found.

CTSPILineConnection::Init

Protected

virtual void Init(CTSPIDevice* *pDevice***, DWORD** *dwLineDeviceID***, DWORD** *dwPosition***);**

pDevice	The device object that owns this line
dwLineDeviceID	The TAPI line device ID associated with this line
dwPosition	The device array position for this line.

Remarks

This method is called directly after the constructor to initialize the line connection object. It is called during the **CServiceProvider::providerInit** method by the device owner (during the **CTSPIDevice::Init**), or when a new line is created using the **CTSPIDevice::CreateLine** method.

If you do not use the SPLUI library to store mini objects in the registry then the service provider should override this method in order to adjust capabilities and add any addresses that are part of the line.

CTSPILineConnection::IsConferenceAvailable

Protected

bool IsConferenceAvailable(CTSPICallAppearance* *pCall***);**

pCall	Call appearance which is looking for a conference.

Remarks

This method determines whether there is a conference call active on the same address and line as the call appearance passed. If the flag **LINEDEVCAPFLAGS_CROSSADDRCONF** is OR'd into the **LINEDEVCAPS.dwDevCapFlags** field, then the method expands the search across all the addresses on the line.

Return Value

TRUE/FALSE if a conference was found on either the same address or the same line (if cross address conferencing is available).

CTSPILineConnection::IsTransferConsultAvailable

Protected

bool IsTransferConsultAvailable(CTSPICallAppearance* *pCall***);**

pCall	Call object to locate consultant call for

Remarks

This determines whether there is a consultant call or some other call that we could transfer to right now. It is used internally by the class library to determine basic call features when call states are changing.

Return Value

TRUE/FALSE if there is an available consultation call available on the same address or line as the given call.

CTSPILineConnection::MakeCall

virtual LONG MakeCall ((DRV_REQUESTID *dwRequestID***,**
HTAPICALL *htCall***, LPHDRVCALL** *lphdCall***, LPCTSTR** *lpszDestAddr***,**
DWORD *dwCountryCode***, LPLINECALLPARAMS const** *lpCallParams***);**

dwRequestID	The TAPI asynchronous request ID to associate with this request.
htCall	The opaque TAPI handle to associate with the created call.
lphdCall	The pointer to an area to store our created call handle.
lpszDestAddr	The dialable number to dial.
dwCountryCode	The country code (zero for the default country).
lpCallParams	Additional call information associated with this request.

Remarks

This method is called in response to an application calling **lineMakeCall**. The **TSPI_lineMakeCall** function must be exported from the provider.

The default behavior is to locate an appropriate address using the **FindAvailableAddress** method. It will then create a call appearance on the address using **CTSPIAddressInfo::CreateCallAppearance**, and finally pass control off to the **CTSPICallAppearance::MakeCall** method to complete the processing of the request.

The final result of this method is that an asynchronous request of type **RTMakeCall** will be generated and placed into the queue detailing the call appearance and dialing information.

Return Value

TAPI error code or the asynchronous request ID if the method was successful in creating the request.

CTSPILineConnection::MSPIdentify

virtual LONG MSPIdentify(GUID* *pGUID***);**

pGUID Returning MSP GUID assignment.

Remarks

This method is called by TAPI 3.0 to find the associated media service provider. It is necessary that the TSP associate the GUID for each line using the **SetMSPGUID** method.

Return Value

TAPI error code or zero if the function was successful.

CTSPILineConnection::OnAddressFeaturesChanged

Protected

virtual DWORD OnAddressFeaturesChanged (CTSPIAddressInfo* *pAddr***, DWORD** *dwFeatures***)**

pAddr The address object which changed its features.
dwFeatures The new **LINEADDRFEATURE_xxx** bits which are valid.

Remarks

This method is called when an address associated with this line object changes its address features. The return value will be used as the final features reported to TAPI.

This method may be overridden to change features reported as necessary.

Return Value

Updated feature flags to report back to TAPI. The default behavior is to return the given **dwFeatures** passed to the method.

CTSPILineConnection::OnCallDeleted

Protected

virtual void OnCallDeleted(CTSPICallAppearance* *pCall***);**

pCalll The call appearance which is being destroyed.

Remarks

This method is called when a call associated with this line object is being deleted due to it being deallocated by all owner/monitor applications. Once this method returns, the call object pointer should be considered invalid and unavailable.

CTSPILineConnection::OnCallFeaturesChanged

Protected

virtual DWORD OnCallFeaturesChanged(CTSPICallAppearance* *pCall*, **DWORD** *dwCallFeatures*);

pCalll	The call appearance which is changing.
dwCallFeatures	The call features which are now in effect for the call.

Remarks

This method is called when a call associated with this line object is changing its feature information in the **LINECALLSTATUS** record. The return value will be used as the final features reported to TAPI.

This method may be overridden to change features reported as necessary.

Return Value

Updated feature flags to report back to TAPI. The default behavior is to return the given **dwCallFeatures** passed to the method.

CTSPILineConnection::OnCallStateChange

Protected

virtual void OnCallStateChange (CTSPIAddressInfo* *pAddr*, **CTSPICallAppearance*** *pCall*, **DWORD** ***dwNewState***, **DWORD** *dwOldState*);

pAddr	The address that the call is associated with.
pCalll	The call appearance which is being destroyed.
dwNewState	The new call state that the call is moving to.
dwOldState	The previous call state for this call.

Remarks

This method is called when a call associated with this line object is changing call states. It is called *after* any of the call features are updated..

CTSPILineConnection::OnLineCapabilitiesChanged

virtual void OnLineCapabiltiesChanged();

Remarks

This method is called internally in the library when any of the information in the **LINEDEVCAPS** structure is changed for any reason. It notifies TAPI using a **LINEDEVSTATE_CAPSCHANGE** notification.

It should be called by the service provider if the **LINEDEVCAPS** structure is modified directly through the **CTSPILineConnection::GetDevCaps** method.

CTSPILineConnection::OnLineFeaturesChanged

Protected

virtual DWORD OnLineFeaturesChanged(DWORD *dwLineFeatures***);**

dwLineFeatures New line features for the line.

Remarks

This method is called when any of the line features are altered on the line by the class library. The return value will be placed into the **LINEDEVSTATUS.dwLineFeatures** field and reported back to TAPI.

Return Value

Updated feature flags to report back to TAPI. The default behavior is to return the given **dwLineFeatures** passed to the method.

CTSPILineConnection::OnLineStatusChange

virtual void OnLineStatusChange (DWORD *dwState***,**
DWORD *dwParam1* **= 0L, DWORD** *dwParam2* **= 0L);**

dwState	The new line state.
dwParam1	Optional parameter depending on the new state.
dwParam2	Optional parameter depending on the new state.

Remarks

This method is used internally in the library when any of the information in the **LINEDEVSTATUS** structure is changed using one of the **CTSPILineConnection** methods or through a child (address or call) object changing. It checks to see if the status change is being monitored by TAPI and then calls TAPI to notify it about the change to the device status if required.

It should be called by the service provider if the **LINEDEVSTATUS** structure is modified directly through the **CTSPILineConnection::GetDevStatus** method.

CTSPILineConnection::OnMediaConfigChanged

virtual void OnMediaConfigChanged();

Remarks

This method is used internally in the library when any of the media configuration information in the **LINEDEVSTATUS** structure is changed. It should be called by the service provider if the **LINEDEVSTATUS** structure is modified directly through the **CTSPILineConnection::GetDevStatus** method.

CTSPILineConnection::OnMediaControl

virtual void OnMediaControl (CTSPICallAppearance* *pCall***,**
DWORD *dwMediaControl***);**

pCall	Call object which is reporting media event
dwMediaControl	Type of media event recorded

Remarks

This method is called by the **CTSPICallAppearance::OnMediaControl** method when a media control event was activated due to a media monitoring event being caught on the passed call.

If media control monitoring is done in the service provider, then either this method or the **CTSPICallAppearance::OnMediaControl** method must be overridden to support it.

CTSPILineConnection::OnPreCallStateChange

Protected

virtual void OnPreCallStateChange (CTSPIAddressInfo* *pAddr***,**
CTSPICallAppearance* *pCall***, DWORD** ***dwNewState*****,**
DWORD *dwOldState***);**

pAddr	The address that the call is associated with.
pCalll	The call appearance which is being destroyed.
dwNewState	The new call state that the call is moving to.
dwOldState	The previous call state for this call.

Remarks

This method is called when a call associated with this line object is changing call states. It is called *before* any of the call features are updated. The class library uses this method to adjust the number of active, onHold, pending and idle call counts.

CTSPILineConnection::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq***, LONG** *lResult***);**

pReq	The request which is being completed.
lResult	The final return code for the request.

Remarks

This method is called when any request that is being managed by the line is completed by the service provider. It gives the line an opportunity to do any cleanup required with the request after it has terminated.

The default implementation of the class library manages several requests:

RTSetTerminal

If the request completes successfully, the line connection will alter the internal terminal map and notify TAPI that the terminal information has changed in the **LINEDEVCAPS** record. This change will be cascaded to the address and calls that are owned by the line.

RTForward

If the request completes successfully, forwarding record(s) will be added to the **LINEDEVCAPS** structure and TAPI will be notified that the information has changed. All addresses on the line will be adjusted as well.

RTCompleteCall

If the request completes successfully, a call completion request will be added to an internal array for notification of the final completion. It may be located using the **CTSPILineConnection::FindCallCompletionRequest** method and deleted using the **CTSPILineConnection::RemoveCallCompletionRequest** method.

RTUncompleteCall

If the request completes successfully, the call completion request that was queued originally will be removed and deleted. TAPI is not notified since this is not considered a *device* request, rather a user request.

CTSPILineConnection::OnRingDetected

void OnRingDetected (DWORD *dwRingMode***, bool** *fFirstRing* **= false);**

dwRingMode	The ring mode which was detected.
fFirstRing	TRUE if this is the first ring.

Remarks

This method should be called by the service provider when a ring event is detected on the line device. It will report the ring event through the **LINEDEVSTATE_RINGING** TAPI event. The ring count is maintained internally based on the ring mode (each ring mode supported on the line has a separate ring counter). The first time ring is detected on the line, the *fFirstRing* parameter should be set to **true** to correctly count the rings.

The method is not called by the library – it is intended to be used by the service provider device code to report ring functionality.

CTSPILineConnection::OnUIDialogClosed

void OnUIDialogClosed(HTAPIDIALOGINSTANCE *htDlgInst***,**
LPVOID *lpItemData***);**

htDlgInst	The UI dialog instance handle.
lpItemDate	Item data associated with the instance.

Remarks

This method is called when the dialog is closed by TAPI. The default implementation does nothing. It can be overridden to react to the closing of the dialog, cleanup resources, etc. The *lpItemData* pointer is the original pointer passed to the **CreateUIDialog** method.

CTSPILineConnection::Open

virtual LONG Open(HTAPILINE *htLine***, LINEEVENT** *lpfnEventProc***,**
DWORD *dwTSPIVersion***);**

htLine	The opaque TAPI handle to assign to the line object.
lpfnEventProc	The callback to associate with this line.
dwTSPIVersion	The version the line negotiated at to shear structures to.

Remarks

This method is called by TAPI when the line device is opened. The default library implementation records the information associated with the line calls the **CTSPIDevice::OpenDevice** method to open the line. If that method fails, **LINEERR_RESOURCEUNAVAIL** is returned, otherwise, a success code is returned.

The **LINEDEVSTATUS** structure is updated to reflect an additional open and TAPI is notified that the **dwNumOpens** field has changed.

Return Value

TAPI error code or zero if the line was opened successfully.

CTSPILineConnection::read

virtual std::istream& read(std::istream& *istm***);**

istm	input iostream to read information from.

Remarks

This method is called during initialization if it is determined that the line object information is contained within the registry. This is only used if the SPLUI library stored object information in the registry.

It may be overridden to read additional information from the stream (other than the information placed there by TSP++).

Note: you *must* retrieve information in the same order as it was stored in the SPLUI user-interface DLL implementation for your provider! If the TSP locks up on loading then check the serialization to ensure you are not reading past the iostream.

CTSPILineConnection::RecalcLineFeatures

virtual void RecalcLineFeatures(bool *fRecalcAllAddresses***=false);**

fRecalcAllAddresses	Whether to call **RecalcAddrFeatures** for each address on this line.

Remarks

This method is called when the line status is changed to re-determine the line's feature set based on the status, address, and call information.

It may be overridden to catch all changes to the line state.

CTSPILineConnection::ReceiveMSPData

virtual LONG ReceiveMSPData(CMSPDriver* *pMSP***,**
CTSPICallAppearance* *pCall***, LPVOID** *lpData***, DWORD** *dwSize***);**

pMSP	Cooresponding MSP driver object
pCall	Call object which is targeted for the media stream
lpData	Block of data from the MSP driver
dwSize	Size of the above block of data.

Remarks

This method is called by TAPI 3.0 when the associated media service provider sends a block of data to the TSP. The default behavior is to invoke the **CTSPICallAppearance::ReceiveMSPData** method on the given call object.

Return Value

TAPI error code or zero if the function was successful.

CTSPILineConnection::RemoveCallCompletionRequest

void RemoveCallCompletionRequest(DWORD *dwCompletionID***,**
bool *fNotifyTAPI* **= FALSE);**

dwCompletionID	The completion ID which is being removed/completed.
fNotifyTAPI	Whether TAPI should be notified about the removal.

Remarks

This method is called to remove a call completion record from the line. For more information on call completions, see the description for **CTSPILineConnection::FindCallCompletion**. If the service provider cancels a completion due to the PBX failing, it should notify TAPI through the **fNotifyTAPI** field. This will cause the class library to issue a **LINEDEVSTATE_COMPLCANCEL** event to TAPI. If the request is canceled through the initial **RTCompleteCall**, then TAPI is not notified of a call completion cancellation.

TAPI is automatically notified that the number of call completions has changed through a **LINEDEVSTATE_NUMCOMPLETIONS** event.

CTSPILineConnection::RemoveTerminal

void RemoveTerminal (int *iTerminalID***);**

iTerminalID	The terminal identifier to remove.

Remarks

This method should be called by the service provider to remove a terminal which was added to the line through the **CTSPILineConnection::AddTerminal** method.

TAPI is notified that the number of terminals has changed through the **LINEDEVSTATE_TERMINALS** event.

This method will cause the terminal identifiers to be adjusted throughout the class library. All objects are notified through the **OnTerminalCountChanged** method and are adjusted accordingly.

CTSPILineConnection::SendDialogInstanceData

void SendDialogInstanceData (HTAPIDIALOGINSTANCE *htDlgInstance***,**
LPVOID *lpParams* **= NULL, DWORD** *dwSize* **= 0L);**

htDlgInstance	Dialog instance to send the data to.
lpParams	Data to send to the dialog.

dwSize	Size of the data block to send to the dialog.

Remarks

This method is used to communicate with a unsolicited user-interface dialog which was created using the **CTSPILineConnection::CreateUIDialog** method. The **lpParams** buffer is automatically thunked to the process where the dialog is.

Note: Embedded pointers are not supported in the **lpParams** buffer.

CTSPILineConnection::Send_TAPI_Event

void Send_TAPI_Event(CTSPICallAppearance* *pCall***, DWORD** *dwMsg***,**
DWORD *dwP1* **= 0L, DWORD** *dwP2* **= 0L, DWORD** *dwP3* **= 0L);**

pCall	Call object to reference in the event.
dwMsg	Event to send to TAPI (**LINE_xxx** message from *tapi.h*).
dwP1	Parameter dependant on the message.
dwP2	Parameter dependant on the message.
dwP3	Parameter dependant on the message.

Remarks

The class library uses this method to notify TAPI about various events happening on the line. It issues events to the line callback that was supplied by TAPI when the line was opened.

It can be called by the service provider to support the various line notifications that are not directly supported by the class library (such as **LINE_DEVSPECIFIC**, or **LINE_DEVSPECIFICFEATURE**).

Most of the notifications which are sent to TAPI are wrapped in other member methods so before using this method directly, check to see if there is a better member method which does the notification already.

CTSPILineConnection::SetBatteryLevel

void SetBatteryLevel (DWORD *dwBattery***);**

dwBattery	Battery level for the cellular or portable device. Should be between zero and 0xffff.

Remarks

This method is used to set the current battery level in the **LINEDEVSTATUS** structure. It will notify TAPI that the battery level has changed through a **LINEDEVSTATE_BATTERY** event.

It is not called internally by the library.

CTSPILineConnection::SetCallHubTracking

virtual LONG SetCallHubTracking(
LPLINECALLHUBTRACKINGINFO *lpTrackingInfo***);**

lpTrackingInfo	Call hub data structure.

Remarks

This method is called by TAPI 3.0 to set the current call hub tracking data for the line. The **TSPI_lineSetCallHubTracking** function must be exported for this to be called.

Return Value

TAPI error code or zero if the function was successful.

CTSPILineConnection::SetDefaultMediaDetection

virtual LONG SetDefaultMediaDetection (DWORD *dwMediaModes***);**

dwMediaModes	Media modes TAPI is interested in.

Remarks

This method is called by TAPI when the media mode(s) being monitored for is changing. The library keeps track of the current requested media modes and implements this method automatically. The current media mode(s) is returned through the **CTSPILineConnection:: GetDefaultMediaDetection** method.

The passed media mode flags are validated against the **LINEDEVCAPS.dwMediaModes** value that is obtained through the media modes supported by all the addresses on the line. If any of the passed media modes are not supported, **LINEERR_INVALMEDIAMODE** is returned.

This function is always called by the TAPI Server as part of the **lineOpen** process for an application which opens the line as an owner. Because of this, TSP++ uses this function to re-notify TAPI about existing call objects which were created while no owner applications were running. This may cause the creation of a secondary thread if TAPISRV is not yet ready to receive call notifications on the line.

Return Value

TAPI error code or zero if the method was successful.

CTSPILineConnection::SetDevConfig

virtual LONG SetDevConfig(TString& *strDeviceClass***,**
const LPVOID *lpDeviceConfig***, DWORD** *dwSize***);**

strDeviceClass	Device class which is being set.
lpDeviceConfig	Device configuration information.
dwSize	Size of the passed device configuration.

Remarks

This method is called in response to an application calling **lineSetDevConfig**. It is used for device-specific extensions that are not provided through the general TAPI interface. It is not directly supported in the class library.

The service provider must override this method in order to provide the device-specific capability. The default implementation of the method returns an error.

Return Value

The class library returns **LINEERR_OPERATIONUNAVAIL**.

CTSPILineConnection::SetDeviceStatusFlags

void SetDeviceStatusFlags (DWORD *dwStatus***);**

dwStatus	Device status flags to set on the line.

Remarks

This method can be called by the service provider to adjust the **LINEDEVSTATUS.dwDevStatusFlags** field. It automatically compares the existing flags with the newly passed flags and notifies TAPI for each change found in the state.

There are member methods to set most of the device status flags – use the **DevStatusConnected**, **DevStatusInService**, **DevLocked**, and **DevMsgWaiting** methods to change the appropriate bits instead of calling this method directly.

CTSPILineConnection::SetLineDevStatus

virtual LONG SetLineDevStatus (DRV_REQUESTID *dwRequestID***,**
DWORD *dwStatusToChange***, bool** *fStatus***);**

dwRequestID	Asynchronous request ID to associate with this request.
dwStatusToChange	Device status flags (**LINEDEVSTATUSFLAGS_xxx**) to set on the line device.
fStatus	TRUE or FALSE to turn the status on or off.

Remarks

This method is called in response to an application calling the **lineSetDevStatus** function. The **TSPI_lineSetDevStatus** function must be exported from the provider. The default implementation validates the **dwStatusToChange** against the **LINEDEVCAPS.dwSettableDevStatus** field and creates a **RTSetLineDevStatus** asynchronous request and inserts it into the queue.

When the service provider processes the request, it should use the appropriate **CTSPILineConnection** methods to change the status of the device so that TAPI is notified correctly.

Return Value

TAPI error code or the asynchronous result code if the request was queued/started.

CTSPILineConnection::SetMediaControl

virtual LONG SetMediaControl (TSPIMEDIACONTROL* *lpMediaControl***);**

lpMediaControl	Media control structure associated with this request.

Remarks

This method is called in response to an application calling the **lineSetMediaControl** function. The **TSPI_lineSetMediaControl** function must be exported from the provider. It is responsible for enabling and disabling control actions on the media stream

associated with this line and all addresses/calls present here. Media control actions can be triggered by the detection of specified digits, media modes, custom tones, and call states. The new specified media controls replace all the ones that were in effect for this line, address, or call prior to this request.

The default behavior of the class library is to invoke the **CTSPIAddressInfo::SetMediaControl** method on each address present on the line.

Return Value

TAPI error code zero if the method was successful.

CTSPILineConnection::SetMSPGUID

void SetMSPGUID(GUID& *guidMSP***);**

guidMSP GUID to represent this MSP device.

Remarks

This method should be called during line initialization to associate the unique identifier for the media service provider with the TSP. This will allow TAPI to locate the proper MSP device when media services are required by the application. The MSP GUID is the COM identifier assigned to the media device.

CTSPILineConnection::SetPermanentLineID

void SetPermanentLineID(DWORD *dwLineID***);**

dwLineID New permanent line identifier for this line.

Remarks

This method is used to associate a unique (across all line devices) identifier with this line device. This identifier may then be used at the device level to quickly locate a line for a received event.

Normally the permanent line identifier would be some switch identifier for the line.

CTSPILineConnection::SetRingMode

void SetRingMode (DWORD *dwRingMode***);**

dwRingMode New ring mode for this line.

Remarks

This method can be called by the service provider to adjust the current ring mode for the line device. It adjusts the **LINEDEVSTATUS.dwRingMode** value and notifies TAPI through a **LINEDEVSTATE_OTHER** event. It is not invoked directly by the class library.

CTSPILineConnection::SetRoamMode

void SetRoamMode (DWORD *dwRoamMode***);**

dwRoamMode New roam mode for this line.

Remarks

This method can be called by the service provider to change the roam mode for the line device. It adjusts the **LINEDEVSTATUS.dwRoamMode** value and notifies TAPI through a **LINEDEVSTATE_ROAMMODE** event. It is not invoked directly by the class library.

CTSPILineConnection::SetSignalLevel

void SetSignalLevel (DWORD *dwSignalLevel***);**

dwSignalLevel New signal level for this line (0 – 0xffff).

Remarks

This method can be called by the service provider to change the signal level for the line device. It adjusts the **LINEDEVSTATUS.dwSignalLevel** value and notifies TAPI through a **LINEDEVSTATE_SIGNAL** event. It is not invoked directly by the class library.

CTSPILineConnection::SetStatusMessages

virtual LONG SetStatusMessages(DWORD *dwLineStates***,**
DWORD *dwAddressStates***);**

dwLineStates **LINEDEVSTATE_xxx** events to route to TAPI.
dwAddressStates **LINEADDRESSSTATE_xxx** events to route to TAPI.

Remarks

This method is called in response to an application calling **lineSetStatusMessages**. The **TSPI_lineSetStatusMessages** function must be exported from the provider. TAPI will keep track of all events being asked for and make a single call to this method with all requested events across all applications. The class library keeps track of this information and references it when notifying TAPI about any changes to the **LINEDEVCAPS** or **LINEDEVSTATUS** structures.

Return Value

TAPI error code zero if the method was successful.

CTSPILineConnection::SetTerminal

virtual LONG SetTerminal(DRV_REQUESTID *dwRequestID***,**
CTSPIAddressInfo* *pAddr***, CTSPICallAppearance*** *pCall***,**
DWORD *dwTerminalModes***, DWORD** *dwTerminalID***, bool** *bEnable***);**

dwRequestID	Asynchronous request ID to associate with this request.
pAddr	Address to set the terminal information for (may be NULL).
pCall	Call to set the terminal information for (may be NULL).
dwTerminalModes	**LINETERMINALMODE_xxx** bits.
dwTerminalID	Terminal identifier to change

bEnable	Whether to enable/disable the terminal.

Remarks

This method is called in response to an application calling **lineSetTerminal**. The **TSPI_lineSetTerminal** function must be exported from the provider. The default implementation of the library is to create and insert a **RTSetTerminal** request into the queue. When the service provider processes this request and completes the request with a zero return code, the class library will store and set the new terminal modes within the library data structures using the **CTSPILineConnection::SetTerminalModes** method.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPILineConnection::SetTerminalModes

void SetTerminalModes (int *iTerminalID***, DWORD** *dwTerminalModes***,**
bool *fRouteToTerminal***);**

iTerminalID	Terminal identifier to adjust (0 to **GetTerminalCount**).
dwTerminalModes	Terminal mode(s) (**LINETERMMODE_xxx**) to adjust.
f*RouteToTerminal*	Whether the terminal is added or removed from modes.

Remarks

This is the method that should be called when a **RTSetTerminal** is completed by the service provider. This stores or removes the specified terminal from the terminal modes given, and then forces it to happen for any existing calls on the line by routing the notification through the **CTSPIAddressInfo::SetTerminalModes**.

TAPI is notified about the change through a **LINEDEVSTATE_TERMINALS** event.

CTSPILineConnection::SetupConference

virtual LONG SetupConference(CTSPICallAppearance* *pCall***,**
DRV_REQUESTID *dwRequestID***, HTAPICALL** *htConfCall***,**
LPHDRVCALL *lphdConfCall***, HTAPICALL** *htConsultCall***,**
LPHDRVCALL *lphdConsultCall***, DWORD** *dwNumParties***,**
const LPLINECALLPARAMS *lpLineCallParams***);**

pCall	Call appearance to setup conference on.
dwRequestID	Asynchronous request id from TAPI.
htConfCall	TAPI's call handle for the conference
lphdConfCall	Returning pointer for TSP conference handle.
htConsultCall	TAPI's call handle for any created consultation call.
lphdConsultCall	Returning pointer for TSP consultation call handle.
dwNumParties	Number of expected parties
lpLineCallParams	Pointer to **LINECALLPARAMS** for new call.

Remarks

This method is called in response to the **TSPI_lineSetupConference** command for a given line device. It validates the parameters, creates the appropriate call appearances and generates a **RTSetupConference** request that will be passed through the derived service provider worker code.

Return Value

TAPI result code

CTSPILineConnection::SetupTransfer

virtual LONG SetupTransfer(CTSPICallAppearance * *pCall*,
DRV_REQUESTID *dwRequestID*, **HTAPICALL** *htConsultCall*,
LPHDRVCALL *lphdConsultCall*,
const LPLINECALLPARAMS *lpCallParams*);

pCall	Call appearance to transfer.
dwRequestID	Asynchronous request id from TAPI.
htConsultCall	TAPI's call handle for any created consultation call.
lphdConsultCall	Returning pointer for TSP consultation call handle.
lpLineCallParams	Pointer to **LINECALLPARAMS** for new call.

Remarks

This method is called in response to the **TSPI_lineSetupTransfer** command for a given line device. It validates the parameters, creates the appropriate call appearances and generates a **RTSetupTransfer** request that will be passed through the derived service provider worker code.

Return Value

TAPI result code

CTSPILineConnection::SupportsAgents

bool SupportsAgents() const;

Remarks

This method returns whether this line has been setup to support agent activity (using **EnableAgentProxy**).

Return Value

TRUE if the line has been registered with **JTAPROXY**. FALSE if not.

CTSPILineConnection::UncompleteCall

virtual LONG UncompleteCall (DRV_REQUESTID *dwRequestID*,
DWORD *dwCompletionID*);

dwRequestID	Request ID which identifies this request.
d*wCompletionID*	Assigned completion identifier.

Remarks

This method is called when an application calls **lineUncompleteCall** to stop a call completion request. The **TSPI_lineUncompleteCall** function must be exported from the provider. The default library behavior is to validate the completion id and create an asynchronous request **RTUncompleteCall** with the original **RTCompleteCall** structure. Once the request is completed by the service provider with a zero return code, the call completion record is removed using **RemoveCallCompletionRequest** automatically.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPILineConnection::ValidateMediaControlList

virtual LONG ValidateMediaControlList(
TSPIMEDIACONTROL* *lpMediaControl***) const;**

lpMediaControl Structure with all the media control information validated.

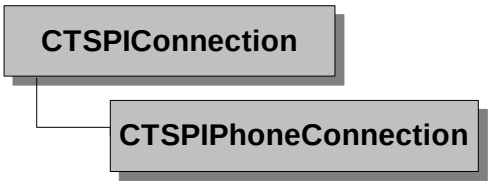
Remarks

This method is called when an application calls **lineSetMediaControl**. It is used to validate that the media control parameters are valid for this line device. It validates the information contained within the structure using the **LINEDEVCAPS** structure and TAPI rules concerning media control lists.

Return Value

TAPI error code or zero if the media control structure is valid.

CTSPIPhoneConnection



A **CTSPIPhoneConnection** object represents a phone device in the library. This object is maintained in a list by the device object. Each created phone device is generally associated with a single line device in the system.

A phone device is defined by the TAPI specification as a device that has some or all of the following capabilities:

Transducer - This is a means for audio input/output. TAPI recognizes that a phone device may have several transducers, which can be taken off-hook or placed on-hook under application or manual user control. TAPI supports three basic types of hook switch devices common to many phone sets:

1. *Handset* - The traditional mouth-and-ear piece combination on most phones.
2. *Speakerphone* - Enables the user to conduct hands-free calls.
3. *Headset* –Standard operator-style headset.

A hook switch must be off-hook to allow audio data to be sent to and/or received by the corresponding transducer.

Volume/Gain/Mute - Each hook switch device is a pairing of a speaker and a microphone. The service provider can provide for volume control and muting of speaker components and for gain control or muting of microphone components if the device supports it.

Ringer - A means for alerting users, usually through a bell or audible tone. A phone device may be able to ring in a variety of modes or patterns.

Display - A mechanism for visually presenting messages to the user.

Buttons - An array of push-buttons on the phone. Button-Lamp IDs identify a button and lamp pair. It is possible to have button-lamp pairs with either no button or no lamp. Button-lamp IDs are values that range from 0 to the maximum number of button-lamps available on the phone device, minus one. Each button belongs to a button class. Button classes include:

1. Call appearances
2. Feature buttons (such as Forward, or Transfer)
3. Digit keypad buttons (0-9, *, #)
4. Local buttons which control speakerphone volume and other phone set options.

Lamps - An array of lamps individually controllable from the service provider. Lamps can be lit in different modes by varying the on and off frequency. The button-lamp ID identifies the lamp. As a lamp changes states, the service provider needs to notify TAPI.

Data Areas - Memory areas in the phone device where instruction code or data can be downloaded to and/or uploaded from.

The TAPI server allows client applications to monitor and control elements of the phone device. The most useful elements for an application are the hook switch devices. The phone set can act as an audio I/O device to the computer, with volume control, gain control, a ringer, data areas (for programming the phone), and a display. Some of these items may not be present, or all of them might be present.

There is no guaranteed core set of services supported by all phone devices. Therefore, the capabilities of the phone device may be very minimal, or have all the TSPI functions supported. These telephony capabilities will vary based on the configuration (client versus client-server or the telephone hardware).

Phone Initialization

During the initialization of a **CTSPIPhoneConnection** object, the initial phone capabilities are setup in the **PHONECAPS** record stored in the object. The capabilities initialized are very minimal, and the derived service provider is responsible for supplying the remainder of the information, either through the **AddXXX** methods, or by adjusting the information in the **PHONECAPS** structure directly. The fields that will most commonly be adjusted will be:

Structure member	Description
dwNumRingModes	This is initialized to one. If more ring patterns are supported on the device, this needs to be set the correct value.

Additional information in the record is set using the **AddUploadBuffer**, **AddDownloadBuffer**, **SetupDisplay**, **AddButton**, **SetPhoneInfo**, and **AddHookSwitchDevice**. These should be called during the phone initialization method or be set into the SPLUI object when it is saved in the registry so that the information is ready when TAPI opens the phone device.

Status Information

Status information for the phone device is stored in an embedded **PHONESTATUS** structure. The library manages all elements of the record. A pointer to this record may be obtained through the **GetPhoneStatus** member method.

Device Specific Extensions

If the phone device supports device-specific extensions to the **PHONECAPS** or **PHONESTATUS** records, then they must be added either by overriding the **CServiceProvider** method **phoneGetDevCaps**, or by overriding the **CTSPIPhoneConnection** method **GatherCapabilities**.

Device specific extension support may be added by overriding the **CServiceProvider::phoneDeviceSpecific** or **CTSPIPhoneConnection::DeviceSpecific** functions.

> **Note:** Do not add the device specific information into the static PHONEDEVCAPS, there is no extra-allocated space!

Constructor and Destructor

CTSPIPhoneConnection	Constructor for the phone object.
~CTSPIPhoneConnection	Destructor for the phone object.

Operations - Public Members

AddButton	This adds a new button or lamp (or both) to the button list maintained in the phone object. The **PHONECAPS** structure will be modified appropriately.
AddDisplayChar	This adds a character to the display at the current cursor position. The cursor position is then adjusted.
AddDisplayString	This adds a string to the display, automatically moving the cursor (both X and Y when necessary).
AddDownloadBuffer	This will add a new download buffer to our phone object. This will adjust the **PHONECAPS** structure appropriately. The download buffer can be manipulated through the **phoneGetPhoneData** method.
AddHookSwitchDevice	This adds a hookswitch device to our list maintained by the phone object. The **PHONECAPS** structure will be modified appropriately.
AddUploadBuffer	This will add a new upload buffer to our phone object. This will adjust the **PHONECAPS** structure appropriately. The upload buffer can be manipulated through the **phoneSetPhoneData** method.
ClearDisplayLine	This clears (blanks with spaces) a line on the display.
FindButtonInfo	Locates a phone button structure (see the reference on *CPhoneButtonInfo*) by function or name.
ForceClose	This forces the phone to be closed immediately.
GetAssociatedLine	Returns the line object associated with this phone.
GetButtonCount	This returns the count of buttons on the phone.
GetButtonInfo	This returns a pointer to the **CPhoneButtonInfo** object that represents a button on the phone. The pointer is declared **const** and therefore cannot be used to change the button information. Use the **SetButtonInfo**

	method for changes to the button.
GetCursorPos	This returns the X/Y position of the cursor on the display. The top-left coordinates are always (0,0).
GetDisplayBuffer	This returns a buffer representing our current display state.
GetLampMode	This returns the lamp mode for a particular lamp/button identifier.
GetPhoneCaps	This returns a pointer to our **PHONECAPS** record. The data may be modified through this pointer.
GetPhoneHandle	Returns the TAPI opaque phone handle associated with this phone device. If the device is not open, (-1) will be returned.
GetPhoneStatus	This returns a pointer to our **PHONESTATUS** record. The data may be modified through this pointer.
ResetDisplay	This method resets the display to spaces, and sets the cursor position to top-left.
Send_TAPI_Event	This method sends an event request to TAPI.
SetButtonInfo	This is called to modify the button information for a button in our button list. This should only be called if the device changes the button information itself (without the service provider). The library will automatically call this if a **RTSetButtonInfo** asynchronous request completes successfully. TAPI is notified through the callback.
SetButtonState	This changes the state of a button in our button list. This should be called if the button state is changed by the device (i.e. a user pressing a function button on the physical phone). TAPI is notified through the callback.
SetDisplay	This allows the service provider to set the entire phone display to a value.
SetDisplayChar	This allows a specific character position on the display to be modified. The cursor position is not changed.
SetDisplayCursorPos	This changes the cursor position to the specified X/Y coordinate location. No data is changed on the display.
SetupDisplay	This allocates a buffer that will be used to represent the phone display. The display is buffered in the phone object so that no request to the phone will be required when TAPI requests the display information. If a display is

	present, this method should be called. The **PHONECAPS** record will be modified appropriately.
SetGain	This changes the microphone gain of a hookswitch device in the **PHONESTATUS** record. This should only be called when the device has changed the hookswitch gain. The library will automatically call this when a **REQUEST_SETHOOKGAIN** completes successfully. TAPI is notified through the callback.
SetHookSwitch	This changes the state of a hookswitch in the **PHONESTATUS** record. This should only be called when the device has changed the hookswitch state. The library will automatically call this when a **RTSetHookswitch** completes successfully. TAPI is notified through the callback.
SetLampState	This changes the state of a lamp in our button list. This should only be called if the lamp state is changed by the device directly. This is automatically called by the library when a **RTSetLampInfo** asynchronous request completes successfully. TAPI is notified through the callback.
SetPermanentPhoneID	Associates the permanent phone identifier with this object for quick lookup at the device level.
SetPhoneFeatures	This changes the **PHONESTATUS** phone features available for TAPI.
SetRingMode	This changes the ring mode in the **PHONESTATUS** record. This should only be called when the device changes the ring mode. The library will automatically call this method when a **RTSetRing** completes. TAPI is notified through the callback.
SetRingVolume	This changes the current ringer volume in the phone object. This should only be called when the device changes the ring volume. The library will automatically call this method when a **RTSetRing** completes. TAPI is notified through the callback.
SetStatusFlags	This changes the **dwStatusFlags** field in the **PHONESTATUS** record. TAPI is notified through the callback.
SetVolume	This changes the volume of a hookswitch device in the **PHONESTATUS** record. This

	should only be called when the device has changed the hookswitch volume. The library will automatically call this when a **RTSetVolume** completes successfully. TAPI is notified through the callback.

Overridables – Public Members

read	Serialization support from the registry.

Overridables - Protected Members

GetRequestMap	Internal method to locate the TSPI request map associated with this phone object. This could be used to provide an object-specific map which is different from the global class-specific request map.
Init	This method is called directly after the constructor to initialize the phone object.
OnButtonStateChange	This method is called when any button changes states (i.e. UP/DOWN). It notifies TAPI about the change through the asynchronous callback.
OnPhoneCapabiltiesChanged	This is called when any information in the **PHONECAPS** record changes.
OnPhoneStatusChange	This is called when any information in our **PHONSTATUS** record changes. It notifies TAPI about the change through the asynchronous callback.
OnRequestComplete	This method is called when any asynchronous request for this phone device has completed. It gives the device an opportunity to examine the return code and change settings in the phone object.

Overridables - TAPI functions

Close	This is called to close the phone device. It will decrement the usage of its parent **CTSPIDevice** object.
DevSpecific	This is called to invoke device-specific commands on the phone in response to a **phoneDevSpecific** request. The default implementation returns **OperationUnavail**.
GatherCapabilities	This method gathers the **PHONECAPS** record for a **phoneGetDevCaps** request. It is handled

	completely by the library.
GatherStatus	This method gathers the **PHONESTATUS** record for a **phoneGetDevStatus** request. It is handled completely by the library.
GenericDialogData	This method is called to handle user-interface dialog data being passed from a UI event.
GetButtonInfo	This method is called in response to a **phoneGetButtonInfo** request. It is handled completely by the library.
GetData	This method is called in response to a **phoneGetData** request. It generates a **RTGetPhoneData** request.
GetDisplay	This method is called in response to a **phoneGetDisplay** request. It is handled completely by the library.
GetGain	This method is called in response to a **phoneGetGain** request. It is handled completely by the library.
GetHookSwitch	This method is called in response to a **phoneGetHookSwitch** request. It is handled completely by the library.
GetLamp	This method is called in response to a **phoneGetLamp** request. It is handled completely by the library.
GetRing	This method is called in response to a **phoneGetRing** request. It is handled completely by the library.
GetVolume	This method is called in response to a **phoneGetVolume** request. It is handled completely by the library.
Open	This is called to open the phone device. It will increment the usage of its parent **CTSPIDevice** object.
SetButtonInfo	This method is called in response to a **phoneSetButtonInfo** request. It generates an asynchronous request for the worker code to complete. If the request completes successfully, the information will be stored back to our button object.
SetData	This method is called in response to a **phoneSetData** request. It generates a **RTSetPhoneData** request.
SetDisplay	This method is called in response to a **phoneSetDisplay** request. It generates a **RTSetDisplay** request.

SetGain	This method is called in response to a **phoneSetGain** request. It generates a **RTSetGain** request.
SetHookSwitch	This method is called in response to a **phoneSetHookSwitch** request. It generates a **RTSetHookswitch** request.
SetLamp	This method is called by the **phoneSetLamp** method to change the status of a lamp indicator. It generates a **RTSetLampInfo** asynchronous request.
SetRing	This method is called in response to a **phoneSetRing** request. It generates a **RTSetRing** request.
SetStatusMessages	This method restricts the messages that will be sent by the library for changes to the **PHONESTATUS** record. It is handled completely by the library.
SetVolume	This method is called in response to a **phoneSetVolume** request. It generates a **RTSetVolume** request.

CTSPIPhoneConnection::AddButton

int AddButton (DWORD *dwFunction*, **DWORD** *dwMode*,
DWORD *dwAvailLampStates*, **DWORD** *dwLampState*,
LPCTSTR *lpszText*);

dwFunction	Button function (**PHONEBUTTONFUNCTION_xxx**).
dwMode	Button mode (**PHONEBUTTONMODE_xxx**).
dwAvailLampStates	Available lamp states (**PHONELAMPMODE_xxx**).
dwLampState	Current lamp state (from available states).
lpszText	ASCII Text for button.

Remarks

This method adds a button to the phone model presented to TAPI. This should be called when the phone is added to the system (such as during **CTSPIPhoneConnection::Init**). Each button added to the phone will be reported through the **PHONEDEVCAPS** and TAPI button information functions.

Return Value

The position of the button within the internal button array.

CTSPIPhoneConnection::AddDisplayChar

void AddDisplayChar (TCHAR *cChar*);

cChar	Character to add to the display.

Remarks

This method adds a new character to the display at the current cursor position. If the provider needs to control the position that the character is inserted at, use the **CTSPIPhoneConnection::SetDisplayChar** method.

CTSPIPhoneConnection::AddDisplayString

void AddDisplayString (LPCTSTR *pszDisplay*);

pszDisplay	NULL terminated string to add to the display.

Remarks

This method adds a new string to the display at the current cursor position. The linefeed character given in the **CTSPIPhoneConnection::SetupDisplay** method is interpreted as a movement to the next row, first column.

CTSPIPhoneConnection::AddDownloadBuffer

int AddDownloadBuffer (DWORD *dwSizeOfBuffer*);

dwSizeOfBuffer	Size of the download buffer to add to the phone.

Remarks

This method adds a download buffer to the phone model presented to TAPI. This should be called when the phone is added to the system (such as during **CTSPIPhoneConnection::Init**). Each buffer added to the phone will be reported through the **PHONEDEVCAPS** structure.

To add a download buffer, the TSP must export either the **TSPI_phoneGetData** or the **TSPI_phoneSetData** function.

Return Value

The position of the buffer within the internal tracking array.

CTSPIPhoneConnection::AddHookSwitchDevice

void AddHookSwitchDevice (DWORD *dwHookSwitchDev***,**
DWORD *dwAvailModes***, DWORD** *dwCurrMode***,**
DWORD *dwVolume* **=-1L, DWORD** *dwGain* **=-1L,**
DWORD *dwSettableModes* **=-1L, DWORD** *dwMonitoredModes* **=-1L);**

dwHookSwitchDev	*D*evice type (**PHONEHOOKSWITCHDEV_xxx**).
dwAvailModes	Hookswitch Modes supported (**PHONEHOOKSWITCHMODE_xxx**).
dwVolume	Current volume (-1 if volume level changes are not supported).
dwGain	Current gain (-1 if gain level changes are not supported).
dwSettableModes	Hookswitch modes which can be set.
dwMonitoredModes	Hookswitch modes which can be monitored.

Remarks

This method adds a hookswitch device to the phone model presented to TAPI. This should be called when the phone is added to the system (such as during **CTSPIPhoneConnection::Init**). Each hookswitch added to the phone will be reported through the **PHONEDEVCAPS** structure.

CTSPIPhoneConnection::AddUploadBuffer

int AddUploadBuffer (DWORD *dwSizeOfBuffer***);**

dwSizeOfBuffer	Size of the upload buffer to add to the phone.

Remarks

This method adds an upload buffer to the phone model presented to TAPI. This should be called when the phone is added to the system (such as during **CTSPIPhoneConnection::Init**). Each buffer added to the phone will be reported through the **PHONEDEVCAPS** structure.

To add an upload buffer, the TSP must export either the **TSPI_phoneGetData** or the **TSPI_phoneSetData** function.

Return Value

The position of the buffer within the internal tracking array.

CTSPIPhoneConnection::ClearDisplayLine

void ClearDisplayLine (int *iRow***);**

iRow Row on the display to clear.

Remarks

This method clears a single row on the display.

CTSPIPhoneConnection::Close()

virtual LONG Close();

Remarks

This method is invoked by **TAPI** when the phone is closed by all applications. It calls the **CTSPIDevice::CloseDevice** method and resets all the phone handle information. Once this method completes, the phone will have no further interaction with TAPI until it is opened again.

Return Value

This method returns a standard TAPI return code.

CTSPIPhoneConnection::CTSPIPhoneConnection

CTSPIPhoneConnection();

Remarks

This is the constructor to the phone object.

CTSPIPhoneConnection::~CTSPIPhoneConnection

~CTSPIPhoneConnection();

Remarks

This is the destructor to the phone object.

CTSPIPhoneConnection::DevSpecific

virtual LONG DevSpecific(DRV_REQUESTID *dwRequestID***,**
LPVOID *lpParams***, DWORD** *dwSize***);**

dwRequestID Asynchronous request ID associated with this request.
lpParams Optional pointer to a parameter block
dwSize Size of the above parameter block

Remarks

This method is called by the library when an application calls **phoneDevSpecific** and identifies this phone object in the **HPHONE** parameter. The **TSPI_phoneDevSpecific**

function must be exported from the provider. This method enables service providers to provide access to features not typically offered by TAPI. The meaning of these extensions are device specific and known only to the service provider and applications which are written to take advantage of them. This method is not handled directly by the library and is provided simply for overriding purposes.

Return Value

You must override this method to provide device-specific functionality. The library returns **PHONEERR_OPERATIONUNAVAIL** if you do not override this method.

CTSPIPhoneConnection::FindButtonInfo

const CPhoneButtonInfo* FindButtonInfo(DWORD *dwFunction***,**
DWORD *dwMode***) const;**
const CPhoneButtonInfo* FindButtonInfo(LPCTSTR *pszText***) const;**

dwFunction	**PHONEBUTTONFUNCTION_xxx** to search for
dwMode	**PHONEBUTTONMODE_xxx** to search for
pszText	Button text to search for

Remarks

This method searches the phone button array for a button that matches the given parameters.

Return Value

A pointer to the button which matches the given criteria, or NULL if no button was found.

CTSPIPhoneConnection::ForceClose

void ForceClose();

Remarks

This method may be called by the service provider to close the phone device immediately regardless of it being in-use or not. It should be used when the device is going offline for some hardware reason or because a component or network connection has been lost.

CTSPIPhoneConnection::GatherCapabilities

virtual LONG GatherCapabilities (DWORD *dwTSPIVersion***,**
DWORD *dwExtVer***, LPPHONECAPS** *lpPhoneCaps***);**

dwTSPIVersion	The TAPI version which structures should reflect.
dwExtVer	The extension version which structures should reflect.
lpPhoneCaps	The structure to fill with phone capabilities.

Remarks

This method is called when an application requests the phone capabilities using the **phoneGetDevCaps** function. The class library automatically fills in all the known

information about the phone using the phone and all the associated button, lamp, and display information.

The service provider can adjust the capabilities returned by using the **GetPhoneCaps** method and modifying the returning structure. This should typically be done when the phone is first initialized as capabilities normally do not change during the life of the provider.

If the structure is modified through the **GetPhoneCaps** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **PHONECAPS** structure.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIPhoneConnection::GatherStatus

virtual LONG GatherStatus (LPPHONESTATUS lpStatus);

lpStatus	The structure to fill with line status.

Remarks

This method is called when an application requests the current line status using the **phoneGetStatus** function. The class library automatically fills in all the known information about the phone using the phone and all the associated button, lamp, and display information.

The service provider can adjust the capabilities returned by using the **GetPhoneStatus** method and modifying the returning structure. Some of the information can be modified using other **CTSPIPhoneConnection** methods. These include **SetButtonInfo**, **SetLampState**, **SetButtonState**, **SetStatusFlags**, **SetRingMode**, **SetRingVolume**, **SetHookSwitch**, **SetVolume**, and **SetGain**. Using these methods is recommended as TAPI is notified about the information changing through a **PHONEDEVSTATUS** callback.

If the structure is modified through the **GetPhoneStatus** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **PHONESTATUS** structure.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIPhoneConnection::GenericDialogData

virtual LONG CTSPIPhoneConnection::GenericDialogData (
LPVOID *lpParam*, **DWORD** *dwSize*);

lpParam	Parameter from the UI dialog.
dwSize	Size of the passed parameter block.

Remarks

This method is called when the user-interface component of the TSP is sending data back to the service provider, and the object type specified in the **TSPI_providerGenericDialogData** function was set to **TUISPIDLL_OBJECT_PHONEID**.

Return Value

Standard TAPI return code. FALSE indicates success.

CTSPIPhoneConnection::GetAssociatedLine

CTSPILineConnection* GetAssociatedLine() const;

Remarks

This method returns the line object which is associated to this phone. Associations are done using the **CTSPIDevice::AssociateLineWithPhone** method.

Return Value

The line object attached to this phone, or NULL if no line is associated to this phone.

CTSPIPhoneConnection::GetButtonCount

int GetButtonCount() const;

Remarks

This method can be used by the service provider to determine the count of buttons which have been added to the phone using the **CTSPIPhoneConnection::AddButton** method.

Return Value

Count of buttons.

CTSPIPhoneConnection::GetButtonInfo

const CPhoneButtonInfo* GetButtonInfo(int *iButtonID***) const;**

iButtonID Button index (zero-based)

Remarks

This method retrieves information about the specified button. The index runs from zero to the count of buttons (see **CTSPIPhoneConnection::GetButtonCount**). The returning object is an internal class library representation of a button.

Return Value

Pointer to a **CPhoneButtonInfo** object which represents this button.

CTSPIPhoneConnection::GetButtonInfo

virtual LONG GetButtonInfo (DWORD *dwButtonId***,**
LPPHONEBUTTONINFO *lpButtonInfo***);**

dwButtonID	Lamp/Button identifier
lpButtonInfo	TAPI **PHONEBUTTONINFO** structure.

Remarks

This method is called when an application calls **phoneGetButtonInfo**. It is responsible for returning the button information for the specified lamp/button identifier. The default implementation returns all the required information from the internal button information object.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetCursorPos

POINT GetCursorPos() const;

Remarks

This method can be used by the service provider to determine where the current cursor position on the display is. This should only be called if a display has been setup by the phone object (see **CTSPIPhoneConnection::SetupDisplay**).

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetData

virtual LONG GetData (DWORD *dwDataID***, LPVOID** *lpData***, DWORD** *dwSize***);**

dwDataID	Identifier of the buffer to retrieve the information from.
lpData	Area to store the buffer into.
dwSize	Size in bytes of the storage area

Remarks

This method is called when an application uses the **phoneGetData** function to retrieve data from a configured phone buffer. The default implementation generates a **RTGetPhoneData** request.

Note that this is a special case since the function is *not* asynchronous according to the TAPI specification. Instead, the library issues the request and then blocks the thread until the request completes.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetDisplay

virtual LONG GetDisplay (LPVARSTRING *lpVarString***);**

lpVarString	Returning data pointer for display.

Remarks

This method is called when an application uses the **phoneGetDisplay** function to retrieve the display information for the phone. The class library fills in the buffer with the current representation of the display based on what has been added/updated using the **AddDisplayChar** and **AddDisplayString** methods.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetDisplayBuffer

TString GetDisplayBuffer() const;

Remarks

This method can be used by the service provider to read the current representation of the display buffer. The returning string is read-only. Any changes made to it will *not* be reflected in the display. If the service provider needs to make changes to the display, it should use the various update methods such as **CTSPIPhoneConnection::AddDisplayChar** and **CTSPIPhoneConnection::AddDisplayString**.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetGain

virtual LONG GetGain (DWORD *dwHookSwitchDevice***,**
LPDWORD *lpdwGain***);**

dwHookSwitchDevice	Hookswitch device to query the gain for.
lpdwGain	Buffer to fill in with current gain value.

Remarks

This method is called when an application uses the **phoneGetGain** function. It is implemented in the class library to retrieve the current gain setting for the specific hook switch device. If the hook switch device doesn't support gain, an error is returned.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetHookSwitch

virtual LONG GetHookSwitch (LPDWORD *lpdwHookSwitch***);**

lpdwHookSwitch	Returning hookswitch values.

Remarks

This method is called when an application uses the **phoneGetHookSwitch** function. It is implemented in the class library to check the **PHONESTATUS. dwHandsetHookSwitchMode**, **PHONESTATUS. dwHeadsetHookSwitchMode**, and **PHONESTATUS.dwSpeakerHookSwitchMode** and return all the hookswitch devices

which are currently active on the phone device. If any of the devices are available and have either MIC, SPEAKER, or both active, then the hookswitch device is returned in the **lpdwHookSwitch** variable.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetIcon

virtual LONG GetIcon (TString& *strDevClass***, LPHICON** *lphIcon***);**

strDeviceClass	Device class which is being asked for.
lphIcon	Returning icon handle.

Remarks

This method is called in response to an application calling **phoneGetIcon**. It is used to return an icon specific to the phone. It is not directly supported in the class library.

The service provider must override this method in order to provide the icon. The default implementation of the method returns an error. Note that TAPI itself will return an icon to the application if the service provider fails the function or does not support it.

Return Value

The class library returns **PHONEERR_OPERATIONUNAVAIL**.

CTSPIPhoneConnection::GetLamp

virtual LONG GetLamp (DWORD *dwButtonId***, LPDWORD** *lpdwLampMode***);**

dwButtonId	Button/Lamp identifier.
lpdwLampMode	Returning **PHONELAMPMODE_xxx** constants.

Remarks

This method is called in response to an application calling **phoneGetLamp**. It is used to return the current state of a specific lamp. The class library implements this using the current setting of the lamps from the internal lamp/button array. The service provider should not call this method, if it needs to know the state of a lamp, use the **CTSPIPhoneConnection::GetLampMode** method.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::GetLampMode

DWORD GetLampMode (int *iButtonId***);**

iButtonId	Button/Lamp identifier.

Remarks

This method can be used by the service provider to determine what the current state of a particular lamp is. It returns the current **PHONELAMPMODE_xxx** constant which reflect the current lamp condition.

Return Value

Current lamp mode for the specified lamp/button identifier.

CTSPIPhoneConnection::GetPhoneCaps

LPPHONECAPS GetPhoneCaps();
const LPPHONECAPS GetPhoneCaps();

Remarks

This method returns the **PHONECAPS** structure which is maintained for the phone and returned by the **GatherCapabilities** method. Any of the non-pointer information may be read or modified through the resulting pointer, although TAPI is not notified if the information is modified directly. To make sure TAPI is correctly updated, use the appropriate member method of the **CTSPIPhoneConnection** object to modify the capabilities of the phone.

WARNING: Do not attempt to add information to the end of the structure, use the provided **CTSPIPhoneConnection** methods to associate the additional pointer information with the phone.

Return Value

Pointer to the **PHONECAPS** structure.

CTSPIPhoneConnection::GetPhoneHandle

HTAPIPHONE GetPhoneHandle() const;

Remarks

This method returns the phone handle which was assigned to the phone connection when it was opened by TAPI.

Return Value

TAPI phone handle or (-1) if the phone is not yet open.

CTSPIPhoneConnection::GetPhoneStatus

LPPHONESTATUS GetPhoneStatus();
const LPPHONESTATUS GetPhoneStatus();

Remarks

This method returns the **PHONESTATUS** structure which is maintained for the phone and returned by the **GatherStatus** method. Any of the non-pointer information may be read or modified through the resulting pointer, although TAPI is not notified if the information is modified directly. To make sure TAPI is correctly updated, use the appropriate member method of the **CTSPIPhoneConnection** object to modify the status of the phone.

If the service provider *does* modify the structure, and wants to notify TAPI, the **CTSPIPhoneConnection::OnPhoneStatusChange** method should be used.

WARNING: Do not attempt to add information to the end of the structure, use the provided **CTSPIPhoneConnection** methods to associate the additional pointer information with the phone.

Return Value

Pointer to the **PHONESTATUS** structure.

CTSPIPhoneConnection::GetRequestMap

Protected

virtual TRequestMap* GetRequestMap() const;

Remarks

This method is called by the library to locate the request map which is used to route TAPI events throught the service provider (see **BEGIN_TSPI_REQUEST** and **END_TSPI_REQUEST**). The default behavior returns the class-global request map.

Return Value

Pointer to the internal request map for this object.

CTSPIPhoneConnection::GetRing

virtual LONG GetRing (LPDWORD *lpdwRingMode***, LPDWORD** *lpdwVolume***);**

lpdwRingMode	Returning ring mode for the phone.
lpdwVolume	Returning ring volume for the phone.

Remarks

This method is called in response to an application calling **phoneGetRing**. The class library implements this method internally by returning the current known ring value. The service provider can adjust this using the **CTSPIPhoneConnection::SetRingMode** and **CTSPIPhoneConnection::SetRingVolume** methods.

Return Value

TAPI error code or zero if successful.

CTSPIPhoneConnection::GetVolume

virtual LONG GetVolume (DWORD *dwHookSwitchDev***,**
LPDWORD *lpdwVolume***);**

dwHookSwitchDev	Hookswitch device to query.
lpdwVolume	Returning speaker volume for the hookswitch.

Remarks

This method is called in response to an application calling **phoneGetVolume**. The class library implements this method internally by returning the current known volume for

the specified hookswitch. The service provider can adjust this using the **CTSPIPhoneConnection::SetVolume** method.

Return Value

TAPI error code or zero if successful.

CTSPIPhoneConnection::Init

Protected

virtual void Init(CTSPIDevice* *pDevice***, DWORD** *dwPhoneDeviceID***, DWORD** *dwPosition***);**

pDevice	The device object that owns this line
dwPhoneDeviceID	The TAPI phone device ID associated with this phone
dwPosition	The position the phone was added to in the device phone array.

Remarks

This method is called directly after the constructor to initialize the phone connection object. It is called during the **CServiceProvider::providerInit** method by the device owner (during the **CTSPIDevice::Init**), or when a new phone is created using the **CTSPIDevice::CreatePhone** method.

If the SPLUI object storage mechanism is not used to configure the service provider, then this method should be overridden in order to adjust capabilities and add any buttons, lamps, hookswitch devices, and display information to the phone.

CTSPIPhoneConnection::OnButtonStateChange

Protected

virtual void OnButtonStateChange (DWORD *dwButtonID***, DWORD** *dwMode***, DWORD** *dwState***);**

dwButtonID	Button identifier which has changed.
dwMode	The mode for the button.
dwState	The new *pressed* state for the button.

Remarks

This method is called when the state of a button changes on the phone. The class library informs TAPI that the button has changed using the **PHONE_BUTTON** event.

CTSPIPhoneConnection::OnPhoneCapabilitiesChanged

Protected

virtual void OnPhoneCapabilitiesChanged();

Remarks

This method is called when the **PHONECAPS** structure is changed. The service provider should also call this method when it modifies the **PHONECAPS** structure for any reason. It is used to notify TAPI.

CTSPIPhoneConnection::OnPhoneStatusChange

Protected

virtual void OnPhoneStatusChange(DWORD *dwState***,**
DWORD *dwParam* **= 0);**

dwState	The phone state which has changed
dwParam	Optional parameter based on the state.

Remarks

This method is called when the state of something has changed on the phone. It should be called by the service provider if something changes on the device and there is not support in the library for it. An example would be device specific extensions.

CTSPIPhoneConnection::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq***, LONG** *lResult***);**

pReq	The request which is being completed.
lResult	The final return code for the request.

Remarks

This method is called when any request which is being managed by the phone is completed by the service provider. It gives the phone an opportunity to do any cleanup required with the request after it has terminated.

The default implementation of the class library manages several requests:

RTSetButtonInfo

If the request completes successfully, the button information is recorded in the internal button structure using the **CTSPIPhoneConnection::SetButtonInfo** method.

RTSetLampInfo

If the request completes successfully, the lamp information is recorded in the internal lamp structure using the **CTSPIPhoneConnection::SetLampState** method.

RTSetRing

If the request completes successfully, the ring information is recorded using the **CTSPIPhoneConnection::SetRingMode** and **CTSPIPhoneConnection::SetRingVolume** methods.

RTSetHookswitch

If the request completes successfully, the hookswitch information is recorded using the **CTSPIPhoneConnection::SetHookSwitch** method.

RTSetGain

If the request completes successfully, the hookswitch information is recorded using the **CTSPIPhoneConnection::SetGain** method.

RTSetVolume

If the request completes successfully, the hookswitch information is recorded using the **CTSPIPhoneConnection::SetVolume** method.

CTSPIPhoneConnection::Open

virtual LONG Open(HTAPIPHONE *htPhone*, **PHONEEVENT** *lpfnEventProc*, **DWORD** *dwTSPIVersion*);

htPhone	The opaque TAPI handle to assign to the phone object.
lpfnEventProc	The callback to associate with this phone.
dwTSPIVersion	The version the phone negotiated at to shear structures to.

Remarks

This method is called by TAPI when the phone device is opened. The default library implementation records the information associated with the phone and calls the **CTSPIDevice::OpenDevice** method to open the phone. If that method fails, **PHONEERR_RESOURCEUNAVAIL** is returned, otherwise, a success code is returned.

Return Value

TAPI error code or zero if the phone was opened successfully.

CTSPIPhoneConnection::read

virtual std::istream& read(std::istream& *istm*);

istm	input iostream to read information from.

Remarks

This method is called during initialization if it is determined that the phone object information is contained within the registry. This is only used if the SPLUI library stored object information in the registry.

It may be overridden to read additional information from the stream (other than the information placed there by TSP++).

Note: you *must* retrieve information in the same order as it was stored in the SPLUI user-interface DLL implementation for your provider! If the TSP locks up on loading then check the serialization to ensure you are not reading past the iostream.

CTSPIPhoneConnection::ResetDisplay

void ResetDisplay();

Remarks

This method can be used by the service provider to clear the display and set it to all blanks. The current cursor position is reset to the (0,0) coordinate.

CTSPIPhoneConnection::Send_TAPI_Event

void Send_TAPI_Event(DWORD *dwMsg*, **DWORD** *dwP1* **= 0L,**
DWORD *dwP2* **= 0L, DWORD** *dwP3* **= 0L);**

dwMsg	Event to send to TAPI (**PHONE_xxx** message from *tapi.h*).
dwP1	Parameter dependant on the message.
dwP2	Parameter dependant on the message.
dwP3	Parameter dependant on the message.

Remarks

This method is used by the class library to notify TAPI about various events happening on the phone. It issues events to the phone callback which was supplied by TAPI when the phone was opened.

It can be called by the service provider to support the various phone notifications which are not directly supported by the class library (such as **PHONE_DEVSPECIFIC**).

CTSPIPhoneConnection::SetButtonInfo

void SetButtonInfo (int *iButtonID*, **DWORD** *dwFunction*, **DWORD** *dwMode*,
LPCTSTR *pszName*);

dwMsg	Event to send to TAPI (**PHONE_xxx** message from *tapi.h*).
dwP1	Parameter dependant on the message.
dwP2	Parameter dependant on the message.
dwP3	Parameter dependant on the message.

Remarks

This method should be called by the provider when information associated with a button changes. It is automatically called by the class library when a **RTSetButtonInfo** completes with a zero return code. The button array object is marked as *dirty* when the button information is changed. This allows the provider code to serialize the array out to some persistent storage if desired.

CTSPIPhoneConnection::SetButtonInfo

virtual LONG SetButtonInfo (DRV_REQUESTID *dwRequestID*,
DWORD *dwButtonId*, **LPPHONEBUTTONINFO const** *lpPhoneInfo*);

dwRequestID	Asynchronous request ID for this request.
dwButtonId	Button identifier to change
lpPhoneInfo	**PHONEBUTTONINFO** structure to set into the button.

Remarks

This method is called when an application uses **phoneSetButtonInfo** to change the information associated with a *soft* button on the phone device. The **TSPI_phoneSetButtonInfo** function must be exported from the provider. It will validate that the button is a valid lamp/button index within the inserted buttons array (see **AddButton**), and then will insert a **RTSetButtonInfo** request into the phone request queue. If the provider completes the request successfully, the **SetButtonInfo** method is

called automatically to change the information within the button array inside the class library.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetButtonInfo** request was inserted into the queue.

CTSPIPhoneConnection::SetButtonState

DWORD SetButtonState (int *iButtonId***, DWORD** *dwButtonState***);**

iButtonID	Button identifier to modify
dwButtonState	New button state (**PHONEBUTTONSTATE_xxx**).

Remarks

This method should be called by the provider when the state of a button changes. It will update the current button information and notify TAPI through a **PHONE_BUTTON** event.

Return Value

Previous button state.

CTSPIPhoneConnection::SetData

virtual LONG SetData (DRV_REQUESTID *dwRequestID***, DWORD** *dwDataID***, LPCVOID** *lpData***, DWORD** *dwSize***);**

dwRequestID	Asynchronous request ID for this request.
dwDataID	Identifier of the buffer to retrieve the information from.
lpData	Data to store into the buffer.
dwSize	Size in bytes of the data.

Remarks

This method is called when an application uses **phoneSetData** to change the information associated with either a download or upload buffer. The **TSPI_phoneSetData** function must be exported from the provider. It validates the buffer identifier against the buffers added using **AddDownloadBuffer** and **AddUploadBuffer**, then inserts a **RTSetPhoneData** request into the phone request queue. The provider is responsible for handling the request – no further work is done in the class library.

Return Value

TAPI error code or the asynchronous request ID if the **PHONE_SETPHONEDATA** request was inserted into the queue.

CTSPIPhoneConnection::SetDisplay

void CTSPIPhoneConnection::SetDisplay (LPCTSTR *pszBuff***);**

pszBuff	Buffer to set as the display.

Remarks

This method is used by the provider to set the current display on the phone. It updates the internal representation of the display and notifies TAPI using a **PHONESTATE_DISPLAY** notification event.

CTSPIPhoneConnection::SetDisplay

virtual LONG SetDisplay (DRV_REQUESTID *dwRequestID***, DWORD** *dwRow***, DWORD** *dwCol***, LPCTSTR** *lpszDisplay***, DWORD** *dwSize***);**

dwRequestID	Asynchronous request ID for this request.
dwRow	Row to set information into
dwCol	Column to set information into
lpszDisplay	Display information to set.
dwSize	Size of the above buffer in bytes.

Remarks

This method is called when an application wants to change the phone display and calls **phoneSetDisplay**. The **TSPI_phoneSetDisplay** function must be exported from the provider. It validates the row and column information against the display device (see **AddDisplay**), and inserts a **RTSetDisplay** into the phone request queue. If the provider completes the request with a zero return code, the internal display information is updated automatically using the **SetDisplay** method.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetDisplay** request was inserted into the queue.

CTSPIPhoneConnection::SetDisplayChar

void SetDisplayChar (int iColumn, int *iRow***, TCHAR** *cChar***);**

iColumn	Column position (X)
iRow	Row position (Y)
cChar	Character to set at the specified position.

Remarks

This method can be used by the service provider to set a single character in the display buffer independent of the current cursor position. The method does not change the current cursor position. It is not used internally in the class library. TAPI is notified using a **PHONESTATE_DISPLAY** event notification.

CTSPIPhoneConnection::SetDisplayCursorPos

void SetDisplayChar (int *iColumn***, int** *iRow***);**

iColumn	Column position (X)
iRow	Row position (Y)

Remarks

This method can be used by the service provider to set the current position of the cursor. This is used for all methods which add characters to the display buffer. TAPI is notified using a **PHONESTATE_DISPLAY** event notification.

CTSPIPhoneConnection::SetGain

void SetGain (DWORD *dwHookSwitchDev***, DWORD** *dwGain***);**

dwHookSwitchDev	Hook-switch device to adjust
dwGain	New gain value for device.

Remarks

This method can be used by the service provider to set the current gain level for a specific hookswitch device TAPI is notified using a **PHONESTATE_HANDSETGAIN**, **PHONESTATE_SPEAKERGAIN**, or **PHONESTATE_HEADSETGAIN** event notification depending on the type of hookswitch device.

CTSPIPhoneConnection::SetGain

virtual LONG SetGain (DRV_REQUESTID *dwRequestId***,**
DWORD *dwHookSwitchDev***, DWORD** *dwGain***);**

dwRequestId	Asynchronous request ID for this request.
dwHookSwitchDev	Hookswitch device to adjust the gain for.
dwGain	New gain level.

Remarks

This method is called when an application wants to change the phone gain for a hookswitch device and calls **phoneSetGain**. The **TSPI_phoneSetGain** function must be exported from the provider. It validates the hookswitch device based on the devices added using **AddHookSwitchDevice**, and inserts a **RTSetGain** into the phone request queue. If the provider completes the request with a zero return code, the internal gain information is updated automatically using the **SetGain** method.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetGain** request was inserted into the queue.

CTSPIPhoneConnection::SetHookSwitch

void SetHookSwitch (DWORD *dwHookSwitchDev***, DWORD** *dwMode***);**

dwHookSwitchDev	Hook-switch device to adjust
dwMode	New setting for the device. (**PHONEHOOKSWITCHMODE_** value)

Remarks

This method can be used by the service provider to set the current state for a specific hookswitch device TAPI is notified using a **PHONESTATE_HANDSETHOOKSWITCH**, **PHONESTATE_SPEAKERHOOKSWITCH**, or **PHONESTATE_HEADSETHOOKSWITCH** event notification depending on the type of hookswitch device.

CTSPIPhoneConnection::SetHookSwitch

virtual LONG SetHookSwitch (DRV_REQUESTID *dwRequestId*, **DWORD** *dwHookSwitchDev*, **DWORD** *dwHookSwitchMode*);

dwRequestID	Asynchronous request ID for this request.
dwHookSwitchDev	Hookswitch device to adjust.
dwHookSwitchMode	New hookswitch state.

Remarks

This method is called when an application wants to change the phone hookswitch state and calls **phoneSetHookSwitch**. The **TSPI_phoneSetHookSwitch** function must be exported from the provider. It validates the hookswitch device based on the devices added using **AddHookSwitchDevice**, and inserts a **RTSetHookswitch** into the phone request queue. If the provider completes the request with a zero return code, the internal hookswitch information is updated automatically using the **SetHookSwitch** method.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetHookswitch** request was inserted into the queue.

CTSPIPhoneConnection::SetLamp

virtual LONG SetLamp (DRV_REQUESTID *dwRequestID*, **DWORD** *dwButtonLampID*, **DWORD** *dwLampMode*);

dwRequestID	Asynchronous request ID for this request.
dwButtonLampID	Button/Lamp identifier
dwLampMode	New lamp mode for the specified lamp id.

Remarks

This method is called when an application wants to change the lamp state and calls **phoneSetLamp**. The **TSPI_phoneSetlamp** function must be exported from the provider. It validates the lamp-id based on the buttons and lamps added using **AddButton**, and inserts a **RTSetLampInfo** into the phone request queue. If the provider completes the request with a zero return code, the internal lamp information is updated automatically using the **SetLampState** method.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetLampInfo** request was inserted into the queue.

CTSPIPhoneConnection::SetLampState

DWORD SetLampState (int *iButtonLampId*, **DWORD** *dwLampState*);

iButtonLampId	Button/Lamp identifier for the lamp to change
dwLampState	New lamp state setting for the device.

Remarks

This method can be used by the service provider to set the current state for a specific lamp on the phone device TAPI is notified using a **PHONESTATE_LAMP** event notification.

CTSPIPhoneConnection::SetPermanentPhoneID

void SetPermanentPhoneID(DWORD *dwPhoneID***);**

dwPhoneID	Unique phone identifer for this device.

Remarks

This method sets the unique (among phone devices) permanent device identifier associated with this phone object. The phone may then be quickly located using this numeric identifier at the device level for event management.

Normally the permanent phone identifier would be some constant identifying the physical unit on the PBX.

CTSPIPhoneConnection::SetPhoneFeatures

void SetPhoneFeatures (DWORD *dwFeatures***);**

dwFeatures	New features for the phone device.

Remarks

This method can be used by the service provider to set the current phone features into the **PHONESTATUS.dwPhoneFeatures** field. TAPI is notified through a **PHONESTATUS_OTHER** event.

CTSPIPhoneConnection::SetRing

virtual LONG SetRing (DRV_REQUESTID *dwRequestID***,**
DWORD *dwRingMode***, DWORD** *dwVolume***);**

dwRequestID	Asynchronous request ID for this request.
dwRingMode	New ringer style (0 – number of ring modes).
dwVolume	New volume level for ringer.

Remarks

This method is called when an application wants to change the ring pattern for the phone and calls **phoneSetRing**. The **TSPI_phoneSetRing** function must be exported from the provider. It validates the ring mode based on the **PHONEDEVCAPS.dwNumRingModes** value, and inserts a **RTSetRing** into the phone request queue. If the provider completes the request with a zero return code, the internal ring-mode information is updated automatically using the **SetRingMode** and **SetRingVolume** methods.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetRing** request was inserted into the queue.

CTSPIPhoneConnection::SetRingMode

void SetRingMode (DWORD *dwRingMode***);**

dwRingMode New ring mode the phone ringer is using.

Remarks

This method can be used by the service provider to set the current ring-mode for the phone ringer device. The ring mode is updated in the **PHONECAPS.dwNumRingModes** field and TAPI is notified using a **PHONESTATE_RINGMODE** event notification.

CTSPIPhoneConnection::SetRingVolume

void SetRingVolume (DWORD *dwRingVolume***);**

dwRingVolume New ring volume the phone ringer is using.

Remarks

This method can be used by the service provider to set the current ring-volume for the phone ringer device. The ring volume is updated in the **PHONESTATUS.dwRingVolume** field and TAPI is notified using a **PHONESTATE_RINGVOLUME** event notification.

CTSPIPhoneConnection::SetStatusFlags

DWORD SetStatusFlags (DWORD *dwStatus***);**

dwStatus Status of the phone device.

Remarks

This method can be used by the service provider to set the current status of the phone device. This method adjusts the **PHONESTATUS.dwStatusFlags** field and then notifies TAPI about what changed. The flags are simply changed to the passed values, so if the provider needs to save what is currently there and *update* it, then it must OR / AND the bits out appropriately.

Based on what changed in the status field, the following notifications are sent:

Operation	Bit field	Notification sent to TAPI
Added	PHONESTATUSFLAGS_CONNECTED	PHONESTATE_CONNECTED
Removed	PHONESTATUSFLAGS_CONNECTED	PHONESTATE_DISCONNECTED
Added	PHONESTATUSFLAGS_SUSPENDED	PHONESTATE_SUSPEND
Removed	PHONESTATUSFLAGS_SUSPENDED	PHONESTATE_RESUME

Return Value

Previous contents of the **PHONESTATUS.dwStatusFlags** field.

CTSPIPhoneConnection::SetStatusMessages

virtual LONG SetStatusMessages(DWORD *dwPhoneStates*, **DWORD** *dwButtonModes*, **DWORD** *dwButtonStates*)

dwPhoneStates	**PHONESTATE_xxx** messages to forward to TAPI.
dwButtonModes	**PHONEBUTTONMODE_xxx** messages to forward to TAPI.
dwButtonStates	**PHONEBUTTONSTATE_xxx** messages to forward to TAPI.

Remarks

This method is called when TAPI wants to change the notification messages that it should receive from the phone device. This adjusts which **PHONE_STATUS** messages are sent to TAPI as internal data structures change. The class library implements this method completely. It is documented so that it may be overridden by the service provider.

Return Value

TAPI error code or zero if the method was successful.

CTSPIPhoneConnection::SetupDisplay

void SetupDisplay (int *iColumns*, **int** *iRows*, **char** *cLineFeed* **=_T('\n'));**

iColumns	Number of columns in display.
iRows	Number of rows in display
cLineFeed	The character which will be used to divide lines.

Remarks

This method creates a new display device on the phone. Only one display device may be created on the phone device, multiple calls to this method will fail. As a result of this method, the **PHONEDEVCAPS** will be altered to show the number of rows/columns available on the phone display, and the **PHONEFEATURE_GETDISPLAY** feature will be added to the **PHONEDEVCAPS.dwPhoneFeatures**. If the **TSPI_phoneSetDisplay** function is exported, the **PHONEFEATURE_SETDISPLAY** feature will also be added.

All display APIs will fail until this method is called to setup the characteristics of the display. You should not call any display methods until this is done.

CTSPIPhoneConnection::SetVolume

void SetVolume (DWORD *dwHookSwitchDev*, **DWORD** *dwVolume*);

dwHookSwitchDev	Hookswitch device to adjust the volume on.
dwVolume	New volume value device is using.

Remarks

This method may be called by the service provider to adjust the internal value for the volume of a hookswitch device on the phone. It adjusts the applicable volume field in the **PHONESTATUS** structure and reports a **PHONESTATE_HANDSETVOLUME**, **PHONESTATE_SPEAKERVOLUME**, or **PHONESTATE_HEADSETVOLUME** notification event back to TAPI based on the hookswitch device that changed.

CTSPIPhoneConnection::SetVolume

virtual LONG SetVolume (DRV_REQUESTID *dwRequestId***,**
DWORD *dwHookSwitchDev***, DWORD** *dwVolume***);**

dwRequestId	Asynchronous request id associated with request.
dwHookSwitchDev	Hookswitch device to adjust the volume for.
dwVolume	New volume level.

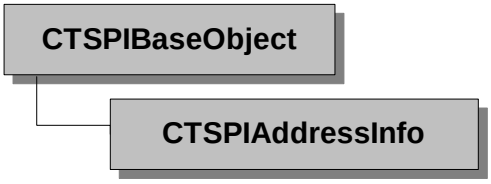
Remarks

This method is called when an application uses the **phoneSetVolume** function to change the volume of a hookswitch device. The provider must export the **TSPI_phoneSetVolume** function. The class library validates the hookswitch device and creates a **RTSetVolume** request and inserts it into the phone request queue. If the provider completes the request with a zero return code, the class library will set the new volume into the **PHONESTATUS** structure using the **SetVolume** method.

Return Value

TAPI error code or the asynchronous request ID if the **RTSetVolume** request was inserted into the phone queue.

CTSPIAddressInfo



An address corresponds directly to a telephone directory number. It is assigned either by the telephone company at the switch (for a POTS or cellular style line), or by the system administrator while configuring the local PBX system.

Addresses are owned by lines, are considered *channels* for the line, and are represented by a **CTSPIAddressInfo** object. This object has a unique dialable address (telephone network address), and a unique number from zero to the total addresses on the line. This number (called an Address ID) is used in conjunction with the line handle to uniquely identify the address to TAPI. Since an address depends on its line to exist, the address's ID is only meaningful in the context of the associated line device. Each **CTSPIAddressInfo** object represents a single address on a line. The address object stores the capabilities, status, and call appearance information that is active for this address. Each request from the TAPI server that is targeted for a specific address will eventually end up in a method of this object.

Address sharing

In general, there is only one address per channel on a line. In the TSP++ library, a channel is generally treated synonymously with an address. Some installations support the assignment of more than one address to a single channel. In a home line configuration (POTS), this is made possible through "distinctive ring" which is an extra-fee service provided by the telephone company. An example of this would be parents who use one address, a child that uses another, and the fax, which uses a third address. When an incoming call appears for any of these addresses, all the phones connected to the line ring, but the ring pattern will be different depending on the number dialed by the calling party. The people in the house can then determine who the incoming call is meant for, and the fax machine answers its calls by recognizing its own ringing pattern.

Many large corporations use DID, or "direct-inward dialing" for incoming calls. Before a call is connected, its destination extension number is signaled to the PBX, which causes the extension to ring instead of the operators phone. In ISDN systems, the various "B" channels might not have separate addresses. Because these "B" channels might be on the same address, it is the service provider that must route the call to the appropriate channel through media detection.

Address Configuration

The relationship of an address to a line and to other addresses is known as its configuration. The network or switch can configure addresses to line assignments in many ways. The types of address configurations that are supported by TAPI are:

Private The address is assigned to one line device only. An inbound call is notified at one line device. This is the default configuration for TSP++.

Bridged A bridged address is a single address assigned to more than one line device. An incoming call will be offered to all lines associated with the address.

Monitored The line indicates the busy or idle status of the address, but no calls may be placed or accepted on the address.

The default configuration for the library is *private*. If this needs to be overridden, then the **ADDRESSCAPS** of the **CTSPIAddressInfo** will need to be adjusted to reflect this. Again, if the channel is shared across addresses, such as a bridged configuration, then special adjustments need to be made. See the above section on *Address Sharing*.

Address initialization

The address objects are created by the **CTSPILineConnection** object in response to the **CreateAddress** method. As part of this method, the line connection owner, address id, dialable address, address name, media modes, bearer modes, data rates, and call appearance information are setup for the address. The main function of the address initialization is to initialize all the fields in the **LINEADDRESSCAPS** and **LINEADDRESSSTATUS** records.

If the object storage mechanism implemented by the SPLUI library is used, then addresses are automatically created based on information in the registry.

Address Capabilities

The capabilities of each address are stored inside the **CTSPIAddressInfo** object. As each address is created and added to a line (through the **CreateAddress** method of the **CTSPILineConnection** object), the passed parameters will give a default set of capabilities to the address. Additionally, if any capabilities need to be adjusted, the **CTSPIAddressInfo** method **GetAddressInfo** will return a pointer to the **ADDRESSCAPS** record where this information is stored. Most of the fields are initialized for the address automatically. The following fields may need to be changed or supplied by the service provider.

Structure member	Description
dwAddressSharing	Store the appropriate LINEADDRESSSHARING_ if not private.
dwSpecialInfo	Report any of the special signaling available on the address.
dwAddrCapFlags	Fill in with the required address capabilities. Default is supplied, but may not be adequate for all systems.
dwRemoveFromConfCaps	If conferencing is supported, this needs to be filled in.
dwRemoveFromConfState	If conferencing is supported, this needs to be filled in.
dwTransferModes	If transfer is supported, this needs to be filled in.
dwParkModes	If call park is supported, this needs to be filled in.
dwForwardModes	If forwarding is supported, this needs to be filled in.
dwMaxForwardEntries	If forwarding is supported, this needs to be filled in.
dwMaxSpecificEntries	If forwarding is supported, this needs to be filled in.
dwMinFwdNumRings	If forwarding is supported, this needs to be filled in.
dwMaxFwdNumRings	If forwarding is supported, this needs to be filled in.
dwMaxCallCompletions	If call completion is supported, this needs to be filled in.
dwCallCompletionConds	If call completion is supported, this needs to be filled in.
dwCallCompletionModes	If call completion is supported, this needs to be filled in.

Address Call Completion

If call completion is supported, then the AddCompletionMessage method should be called to add all the completion messages supported for display on the line. This will fill out the appropriate fields in the LINEADDRESSCAPS structure.

Maximum call counts

If any of the **dwMaxNum** call fields in the **LINEADDRESSCAPS** record are changed, then the **dwMaxNumActiveCalls** in the **CTSPILineConnection LINEDEVCAPS** may also need to be adjusted. This cannot done automatically after the address is created since there is no way to detect modifications done through the **GetAddressInfo** method of **CTSPIAddressInfo**.

Device Specific Extensions

If the address supports device-specific information being returned in the **LINEADDRESSCAPS** record, then these must be added either by overriding the **CServiceProvider** method **lineGetAddressCaps**, or by overriding the **CTSPIAddressInfo** method **GatherCapabilities.** If the address supports this extension information for the **LINEADDRESSSTATUS** record, then it must be added either by overriding the **CServiceProvider** method **lineGetAddressStatus**, or by overriding the **CTSPIAddressInfo** method **GatherStatusInformation.** Also, the **dwAddressStates** member of the **LINEADDRESSCAPS** structure will need to have the **LINEADDRESSSTATE_DEVSPECIFIC** bit flag added to it.

Note: Do not add the device specific information into the static LINEADDRESSCAPS, there is no extra-allocated space!

Address Status

The state of the address is managed completely by the **CTSPIAddressInfo** object. The TAPI server expects the status of an address to be returned in a **LINEADDRESSSTATUS** structure during a call to **lineGetAddressStatus**. A pointer to the structure maintained by the library may be retrieved using the **GetAddressStatus** method. All fields except the device-specific information are filled out.

Request completions

When a request completes on the line, the **OnRequestComplete** method is called. The address object takes the following actions when processing requests:

RTSetTerminal If the request completes successfully, the address will alter the internal terminal map and notify TAPI that the terminal information has changed in the **LINEADDRESSCAPS** record. This change will be cascaded to the calls which are owned by the address.

RTForward If the request completes successfully, the address will add a forwarding record to the **LINEADDRESSCAPS** record and notify TAPI that forwarding information has changed. If the request failed, then the consultation call created will be destroyed.

Constructor and Destructor

CTSPIAddressInfo	Constructor for the address object
~CTSPIAddressInfo	Destructor for the address object

Operations - Public Members

AddAgentGroup	Adds a new agent group to the address – effectively logging the agent into the group (according to TAPI).
AddCallTreatment	Stores a call treatment information block with the **LINEADDRESSCAPS** structure.
AddCompletionMessage	Add a completion message (*initialization only)*
AddDeviceClass	Adds a new device class object to the **LINEADDRESSCAPS** structure.
AddForwardEntry	Manually add a forwarding request into this address. This should be used at initialization if the device is *already* forwarded to some destination so that the class library may be kept in synch with the device.
CanAnswerCalls	Returns TRUE/FALSE if this address can answer offering calls. This information is provided when the address is added to the line.

CanMakeCalls	Returns TRUE/FALSE if this address can ever make outgoing calls. This information is provided when the address is added to the line.
CreateCallAppearance	Create a new call appearance on this address
CreateConferenceCall	Create a new conference call appearance on this address.
FindAttachedCall	This locates a call attached to the specified call on this address.
FindCallByCallID	Find a call appearance based on the **dwCallID** field of the **LINECALLINFO** record.
FindCallByHandle	Find a call appearance based on an opaque TAPI handle.
FindCallByState	Find a call appearance based on a **LINECALLSTATE_**
GetAddressCaps	This returns a pointer to the **LINEADDRESSCAPS** record. The information within the record may be changed.
GetAddressID	This returns the Address ID assigned to this address. It is a value from zero to the number of addresses on the line minus one.
GetAddressStatus	This returns a pointer to the **LINEADDRESSSTATUS** record. The information within the record may be changed.
GetAddressType	TAPI 3.0 function to retrieve the type of address this object represents (IP, phone, domain, etc.)
GetAgentCaps	Returns a pointer to the agent capabilities structure.
GetAgentStatus	Returns a pointer to the agent status structure.
GetAvailableMediaModes	This returns the media modes supported on this address. The available media modes are set by the **CTSPILineConnection::CreateAddress** method.
GetBearerMode	Returns the current bearer mode of this address. This is set during the creation of the address in the **CTSPILineConnection::CreateAddress** method.
GetCallCount	This returns the current count of calls on this address.
GetCallInfo	Return the **CTSPICallAppearance** for a call based on an index.
GetCallTreatmentName	Returns a call treatment block from the **LINEADDRESSCAPS** structure.
GetCompletionMessageCount	Return the number of completion messages
GetCompletionMessage	Return a completion message based on an index.
GetCurrentAgentGroup	Returns a logged on group by array index.
GetCurrentAgentGroupCount	Returns the current number of groups this address is currently a logged into. This is *not* the list of available groups.
GetCurrentAgentState	Returns the current agent state for this address.
GetCurrentRate	This returns the current data rate for the address. This

	is set during the creation of the address in the **CTSPILineConnection::CreateAddress** method.
GetDeviceClass	Returns a device class object from the **LINEADDRESSCAPS** structure.
GetDialableAddress	This returns the dialable telephone network address assigned to this object.
GetLineOwner	This returns a pointer to the owning **CTSPILineConnection** object.
GetName	This returns the address name. This name will be used as the caller-id name when outgoing calls are placed on this address.
MoveCall	Moves a call appearance from another address to this one. Effectively copies the existing call information into a new object owned by the current address and then idles the original call.
RemoveAllAgentGroups	Clears the current list of agent groups this address is logged into.
RemoveCallAppearance	Remove and delete a call appearance.
RemoveCallTreatment	Removes a call treatment record from the **LINEADDRESSCAPS** structure.
RemoveDeviceClass	Removes a device class from the ADDRESSCAPS structure.
SetAddressFeatures	This is used to change the current address features reported to TAPI.
SetAgentActivity	Changes the current reported activity for this address/agent.
SetAgentFeatures	Changes the current list of agent features associated with the current address.
SetAgentGroup	Logs the current address into a list of groups (used automatically by TSP++ when a **lineSetAgentGroup** request completes successfully).
SetAgentState	Sets the current and next agent states.
SetCurrentRate	This sets the current data rate reported on the address. This new rate will then be used as the maximum rate for any new calls created on the address.
SetDialableAddress	This changes the dialable address associated with this object which is reported as caller-id information.
SetMediaControl	This method is called to enable and disable media control actions. It is called when a **lineSetMediaControl** completes successfully.
SetName	This changes the name associated with the address which is reported as caller-id information.
SetNumRingsNoAnswer	Sets the number of rings before considered a “no answer” condition. This is stored in the **LINEADDRESSSTATUS** record.
SetTerminalModes	This method sets the terminal routing information.

	This is called by the library when a **RTSetTerminal** completes successfully.
SetValidAgentStates	Changes the valid states that may be used with **lineSetAgentState** function.
SetValidNextAgentStates	Changes the valid next states that may be used with the **lineSetAgentState** function.

Overridables - Public Members

CanForward	Validates a forwarding information request for this address.
CanSupportCall	Returns whether the call type can be carried on this address.
CanSupportMediaModes	Called to determine if a particular set of media modes can be supported on this address
OnAddressCapabilitiesChange	This should be called by the service provider if any of the fields in the **LINEADDRESSCAPS** record change after the service provider has been initialized.
OnAddressStateChange	This method is called when any of the data in our **LINEADDRESSSTATUS** record has changed.
read	Serialization method for the registry

Operations - Protected Members

AddAsynchRequest	This queues a request into the device asynchronous list.
DeleteForwardingInfo	This method is used to delete the forwarding information associated with this address.
GetTerminalInformation	Returns the terminal information for this address.

Overridables - Protected Members

Init	This method is called directly after the constructor to initialize the address object.
NotifyInUseZero	This method is called to check and send the **LINEADDRESSTATUS_INUSEZERO** notifcation to TAPI.
OnAddressFeaturesChanged	This method is used to adjust the address features associated with this address and allow the service provider to change them before TAPI is notified.
OnAgentCapabilitiesChanged	This method is called when the agent capabilities have changed on this address.
OnAgentStatusChanged	This method is called when the agent status for this address has changed.
OnCallFeaturesChanged	This method is called by the call appearance if it changes its features.

OnCallStateChange	Called by call appearances owned by this address when any call state information changes after TAPI has been notified.
OnCreateCall	This method is called when any new call appearance is created on this address.
OnPreCallStateChange	This method is called right before TAPI is notified about a call state change on this address. It is called by the call appearance.
OnRequestComplete	This method is called when any request completes which was associated with this address.
OnTerminalCountChanged	This method is called by the line owner when any of the terminal counts change due to a **AddTerminal** or **RemoveTerminal** method call in the **CTSPILineConnection**.
RecalcAddrFeatures	This is called to recalculate the address features according to the calls and current state of the line device.

Overridables - TAPI Members

AgentSpecific	This method is called for **lineAgentSpecific**. The default implementation issues a **RTAgentSpecific** request.
CloseMSPInstance	TAPI 3.0 method which is called in response to the **TSPI_lineCloseMSPInstance** function.
CompleteTransfer	This method is called for **lineCompleteTransfer**. The default implementation issues a **RTCompleteTransfer** request.
CreateMSPInstance	TAPI 3.0 method which is called in response to the **TSPI_lineCreateMSPInstance** function.
DevSpecific	This method is called for **lineDevSpecific**. Default implementation is to report **Not Supported**.
Forward	This method is called for **lineForward**. If the request completes successfully, it will update the forwarding information for the address automatically.
GatherAgentCapabilities	This method is called for the **lineGetAgentCaps** function. It is handled completely within the library.
GatherAgentStatus	This method is called for the **lineGetAgentStatus** function. It is handled completely within the library.
GatherCapabilities	This method is called for the **lineGetAddressCaps** function. It is handled completely within the library.
GatherStatusInformation	This method is called for **lineGetAddressStatus**. It is handled completely within the library.
GetAgentActivityList	This method is called for the **lineGetAgentActivityList** function. It is handled completely within the library.

GetAgentGroupList	This method is called for the **lineGetAgentGroupList** function. It is handled completely within the library.
GetID	This method is called for **lineGetID**. Default implementation is to return information in the device class array which was added with **AddDeviceClass**.
Pickup	This method is called for **linePickup**. The default implementation issues a **RTPickup** request.
SetAgentActivity	This method is called for **lineSetAgentActivity**. The default implementation issues a **RTSetAgentActivity** request.
SetAgentGroup	This method is called for **lineSetAgentGroup**. The default implementation issues a **RTSetAgentGroup** request.
SetAgentState	This method is called for **lineSetAgentState**. The default implementation issues a **RTSetAgentState** request.
SetStatusMessages	This sets the status messages TAPI wishes to be notified about on this address. This is called by the line owner when a **lineSetStatusMessages** is received.
SetupConference	This method is called for **lineSetupConference**. The default implementation creates the conference handles and issues a **RTSetupConference** request.
SetupTransfer	This method is called for **lineSetupTransfer**. The default implementation issues a **RTSetupTransfer** request.
Unpark	This method is called for **lineUnpark**. The default implementation issues a **RTUnpark** request.

CTSPIAddressInfo::AddAgentGroup

void AddAgentGroup(DWORD *dwGroupID1***, DWORD** *dwGroupID2***=0, DWORD** *dwGroupID3***=0, DWORD** *dwGroupID4***=0, LPCTSTR** *pszGroupName***=NULL);**

dwGroupID1	First 32-bit value of the 128-bit unique group identifier.
dwGroupID2	Second 32-bit value of the 128-bit unique group identifier.
dwGroupID3	Third 32-bit value of the 128-bit unique group identifier.
dwGroupID4	Fourth 32-bit value of the 128-bit unique group identifier.
pszGroupName	Text name of the group

Remarks

This method adds a new group to the current logged on list of groups for this address. The group should be one of the available groups in the device object (see the reference section on *CTSPIDevice*) although this isn't a requirement.

This method *does not* notify TAPI that the agent status has changed – after calling this method, call the **OnAgentStatusChanged** method. This is not done so that the groups may be completely changed before notifying TAPI, for example if the current groups need to be removed or there is more than one group to add.

CTSPIAddressInfo::AddAsynchRequest

Protected

int AddAsynchRequest(CTSPIRequest* *pReq***);**

pReq	Request object to insert into the line request list.

Remarks

This method inserts a new request into the pending line request list. The **CTSPIRequest** object is inserted into the list associated with the owning **CTSPILineConnection** object. If there are no running requests in the list, then this request is actually started and given the opportunity to run the **STATE_INITIAL** processing before this method returns

Return Value

Position where the request was inserted in the list.

CTSPIAddressInfo::AddCallTreatment

void AddCallTreatment (DWORD *dwCallTreatment***, LPCTSTR** *pszName***);**

dwCallTreatment	Call treatment type
pszName	ASCII Name of the call treatment.

Remarks

This method creates an entry in the Call Treatment handler for this address. The call treatment entry will be returned in the **LINEADDRESSCAPS** structure. The call

treatment type can either be one of the TAPI-specified entries (**LINECALLTREATMENT_xxx**), or it can be a service-provider defined value.

The current call treatment indicates the sounds a party on a call that is unanswered or on hold hears.

CTSPIAddressInfo::AddCompletionMessage

int AddCompletionMessage (LPCTSTR *pszBuff***);**

pszBuff	Completion message to add.

Remarks

This method creates an entry in the Call Completion Message handler for this address. The call completion entry will be returned in the **LINEADDRESSCAPS** structure.

Return Value

Position within the call completion message array for this entry. The array is sequential, starting at zero.

CTSPIAddressInfo::AddDeviceClass

int CTSPIConnection::AddDeviceClass (LPCTSTR *pszClass***,**
DWORD *dwData***);**

int CTSPIConnection::AddDeviceClass (LPCTSTR *pszClass***,**
HANDLE *hHandle***, LPCTSTR** *lpszBuff***);**

int CTSPIConnection::AddDeviceClass (LPCTSTR *pszClass***,**
HANDLE *hHandle***, LPVOID** *lpBuff***, DWORD** *dwSize***);**

int CTSPIConnection::AddDeviceClass (LPCTSTR *pszClass***,**
DWORD *dwFormat***, LPVOID** *lpBuff***, DWORD** *dwSize***,**
HANDLE *hHandle* **= INVALID_HANDLE_VALUE);**

int CTSPIConnection::AddDeviceClass (LPCTSTR *pszClass***,**
LPCTSTR *pszBuff***, DWORD** *dwType* **= -1L);**

pszClass	Device class to add/update for (e.g. “tapi/line”, etc.)
dwData	DWORD data value to associate with device class.
hHandle	Win32 handle to associate with device class.
lpszBuff	Null-terminated string to associate with device class.
lpBuff	Binary data block to associate with device class
dwSize	Size of the binary data block
dwType	**STRINFORMAT** of the **lpszBuff** parameter.

Remarks

This method adds an entry to the device class list, associating it with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEADDRESSCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

Index array of added structure.

CTSPIAddressInfo::AddForwardEntry

int AddForwardEntry (DWORD *dwForwardMode***, LPCTSTR** *pszCaller***,**
LPCTSTR *pszDestination***, DWORD** *dwDestCountry***);**

dwForwardMode	Forwarding mode for the address.
pszCaller	Caller to be forwarded.
pszDestination	Destination to forward this address to.
dwDestCountry	Country code to forward to.

Remarks

This method allows the direct addition of forwarding information on this address. This should only be used if the service provider can detect that the address is already forwarded on initialization.

If the service provider is forwarding the phone due to TAPI requesting the forward, this method is called automatically by the class library when the **RTForward** is completed. This method should only be called by the service provider to adjust the forwarding information for existing forwarding conditions, or due to unsolicited forwarding requests by the phone device or switch.

To delete the forwarding information, pass a zero in for forward mode.

Return Value

Index array of forwarding information.

CTSPIAddressInfo::AgentSpecific

virtual LONG AgentSpecific(DRV_REQUESTID *dwRequestID***,**
DWORD *dwAgentExtensionIDIndex***, LPVOID** *lpvBuff***, DWORD** *dwSize***);**

dwRequestID	TAPI request id for asynchronous operation
dwAgentExtensionIDIndex	Agent extension id being requested.
lpvBuff	Parameter buffer for request.
dwSize	Size of parameter buffer

Remarks

This method is called when TAPI invokes the **lineAgentSpecific** function. It validates the parameters and then generates a **RTAgentSpecific** request and queues it to the line object for handling.

This method will only be called if the address adds device specific extensions to the agent support through the device object's **AddAgentSpecificExtension** method.

Return Value

TAPI result code

CTSPIAddressInfo::CanAnswerCalls

bool CanAnswerCalls() const;

Remarks

This method returns the value given to the **CTSPIAddressInfo::Init** method regarding whether the address is capable of receiving incoming calls or not. This will almost always be **TRUE** unless the address is a *dial-out* address only.

CTSPIAddressInfo::CanForward

virtual LONG CanForward(TForwardInfoArray* *parrForwardInfo***,**
LPDWORD *pdwNumRings***, int** *iCount***);**

parrForwardInfo	Array of **TSPIFORWARDINFO** structures from a **lineForward** call.
pdwNumRings	Number of rings to wait before forwarding.
iCount	Number of addresses being forwarded (zero is all).

Remarks

This method is called to verify that this address can forward given the specified forwarding information. All addresses being forwarded in a group will be given a chance to check the forwarding request before the **Forward** method is actually invoked to insert the asynchronous request.

Return Value

TAPI error code or zero if the forwarding information is acceptable for the address.

CTSPIAddressInfo::CanMakeCalls

bool CanMakeCalls() const;

Remarks

This method returns the value given to the **CTSPIAddressInfo::Init** method regarding whether the address is capable of creating outgoing calls or not. This will almost always be **TRUE** unless the address is hardwired to receive calls only.

CTSPIAddressInfo::CanSupportCall

virtual LONG CanSupportCall (
const LPLINECALLPARAMS *lpCallParams***) const;**

lpCallParams	**LINECALLPARAMS** structure to validate.

Remarks

This method checks the contents of the **LINECALLPARAMS** structure and returns whether or not the call can be supported on this address. The media-mode and call parameter flags are checked against the **LINEADDRESSCAPS** fields.

Return Value

TAPI error code if the call cannot be supported indicating reason.

CTSPIAddressInfo::CanSupportMediaModes

virtual bool CanSupportMediaModes (DWORD *dwMediaModes***) const;**

dwMediaModes	**LINEMEDIAMODE_xxx** Media mode flags to validate.

Remarks

This method returns whether or not the specified media modes are supported on this address. This is done by using the values given to the address when it was created.

Return Value

TRUE or FALSE whether the specified media modes are valid for this address.

CTSPIAddressInfo::CloseMSPInstance

virtual LONG CreateMSPInstance(CMSPDriver* *pMSP***);**

pMSP	Pointer to the MSP driver instance created earlier.

Remarks

This TAPI 3.0 method closes the media service provider instance opened by **CreateMSPInstance**. It is called in response to **TSPI_lineCloseMSPInstance**. Once this method completes, the MSP handle will no longer be valid.

Return Value

TAPI error code or zero if successful.

CTSPIAddressInfo::CompleteTransfer

virtual LONG CompleteTransfer (DRV_REQUESTID *dwRequestId***,**
CTSPICallAppearance* *pCall***, CTSPICallAppearance*** *pConsult***,**
HTAPICALL *htConfCall***, LPHDRVCALL** *lphdConfCall***,**
DWORD *dwTransferMode***);**

dwRequestId	Asynchronous request ID associated with this request.
pCall	Target of the transfer event.
pConsult	Destination of the transfer target.
htConfCall	TAPI opaque handle if the transfer results in a conference.
lphdConfCall	Return pointer for TSP handle to resulting conference.
dwTransferMode	**LINETRANSFERMODE_xxx** of the type of transfer to complete.

Remarks

This method is called when an application completes a conference using **lineCompleteTransfer**. The TSP must export the **TSPI_lineCompleteTransfer** function. Some telephony devices allow a transfer to complete into a three-way conference call, for this reason, TAPI sends a call handle to use if this happens. The class library validates that the call is in a valid call state, validates the transfer type (to conference or regular) based on the **LINEADDRESSCAPS**, creates a conference call appearance if the transfer is moving into a conference, and finally, submits a **RTCompleteTransfer** asynchronous request into the request queue.

Return Value

TAPI error code or the asynchronous request ID if a **RTCompleteTransfer** request was placed into the request queue.

CTSPIAddressInfo::CreateCallAppearance

CTSPICallAppearance* CreateCallAppearance(HTAPICALL *htCall* **= NULL,**
DWORD *dwCallParamFlags* **= 0,**
DWORD *dwOrigin* **= LINECALLORIGIN_UNKNOWN,**
DWORD *dwReason* **= LINECALLREASON_UNKNOWN,**
DWORD *dwTrunk* **= 0xffffffff, DWORD** *dwCompletionID* **= 0);**

htCall	TAPI opaque handle to represent this call (**NULL** if the call is being generated by the TSP. This will cause the TSP to ask TAPI for a new call handle).
dwCallParamFlags	Initial **LINECALLINFO.dwCallParamFlags** settings.
dwOrigin	Initial **LINECALLORIGIN_xxx** value.
dwReason	Initial **LINECALLREASON_xxx** value.
dwTrunk	Initial external trunk which was seized for this call.
dwCompletionID	Completion identifier for a newly completed call. This should match the original completion request ID given by TAPI when the **lineCompleteCall** function was called.

Remarks

This method creates a new call appearance on the address. The appropriate data structures are established, and the **LINEADDRESSSTATUS** fields are updated to reflect a new call. The call state starts out in the **LINECALLSTATE_UNKNOWN** state until the first time it is changed by the service provider. If a call appearance using the specified **htCall** parameter already exists, it is returned. If the **htCall** parameter is NULL, then a new call appearance is established in TAPI using a **LINE_NEWCALL** event.

Return Value

New call appearance pointer, NULL if the call could not be created.

CTSPIAddressInfo::CreateConferenceCall

CTSPICallAppearance* CreateConferenceCall(HTAPICALL *htCall* **= NULL);**

htCall	TAPI opaque handle to represent this call (**NULL** if the call is being generated by the TSP. This will cause the TSP to ask TAPI for a new call handle).

Remarks

This method creates a new conference call appearance on the address. The appropriate data structures are established, and the **LINEADDRESSSTATUS** fields are updated to reflect a new call. If a call appearance using the specified **htCall** parameter already exists, it is returned. If the **htCall** parameter is NULL, then a new call appearance is

established in TAPI using a **LINE_NEWCALL** event. The bearer mode, rate, origin, reason, and trunk are all established from the values in the parent address object.

Return Value

New conference call appearance pointer, NULL if the call could not be created.

CTSPIAddressInfo::CreateMSPInstance

virtual LONG CreateMSPInstance(HTAPIMSPLINE *htMSPLine***,**
LPHDRVMSPLINE *lphdMSPLine***)**

htMSPLine	Handle the MSP driver instance
lphdMSPLine	Pointer to fill in with the TSP-assigned handle.

Remarks

This TAPI 3.0 method creates a new association for a link to a media service provider to handle media services for this TSP. This function creates an associated **CMSPDriver** object and adds it to the MSP map for the given address. The returning *lphdMSPLine* address is filled with a unique TSP-generated identifier which can be used to find the MSP driver object later. This is called in response to **TSPI_lineCreateMSPInstance**.

Return Value

TAPI error code or zero if successful.

CTSPIAddressInfo::CTSPIAddressInfo

CTSPIAddressInfo();

Remarks

This is the constructor for the address object. It should only be called by the class library.

CTSPIAddressInfo::~CTSPIAddressInfo

virtual ~CTSPIAddressInfo();

Remarks

This is the destructor for the address object. It should only be called by the class library. It may be overridden to delete any additional data added to the object.

CTSPIAddressInfo::DeleteForwardingInfo

void DeleteForwardingInfo();

Remarks

This method is used to delete the information in the address objects forwarding array. This array holds the forwarding information reported to TAPI during any

lineGetAddressStatus. This method is called during the destruction of the object and when new forwarding information replaces the existing information.

CTSPIAddressInfo::DevSpecific

virtual LONG DevSpecific(CTSPICallAppearance* *pCall***,**
DRV_REQUESTID *dwRequestID***, LPVOID** *lpParam***, DWORD** *dwSize***);**

pCall	Call this request is for (may be NULL).
dwRequestID	Asynchronous request ID associated with this request.
lpParams	Optional pointer to a parameter block
dwSize	Size of the above parameter block

Remarks

This method is called by the library when an application calls **lineDevSpecific** and identifies this address object in the **dwAddressID** parameter. The **TSPI_lineDevSpecific** function must be exported from the provider. This method enables service providers to provide access to features not typically offered by TAPI. The meaning of these extensions are device specific and known only to the service provider and applications which are written to take advantage of them. This method is not handled directly by the library and is provided simply for overriding purposes.

Return Value

You must override this method to provide device-specific functionality. The library returns **LINEERR_OPERATIONUNAVAIL** if you do not override this method.

CTSPIAddressInfo::FindAttachedCall

CTSPICallAppearance* FindAttachedCall (
CTSPICallAppearance* *pSCall***) const;**

psCall	Source call.

Remarks

This method returns a pointer to the call which is attached to the specified source call. This allows a chain of attached calls. This feature is primarily used to track consultation calls and conferences internally in the library. Generally, the attachment is *one-way*, i.e. the attached call is not linked back to its attached call.

Return Value

Call which is *attached* to the given call object. NULL if no call is attached.

CTSPIAddressInfo::FindCallByCallID

CTSPICallAppearance* FindCallByCallID (DWORD *dwCallID***) const;**

dwCallID	Call Identifier to search the call list for.

Remarks

This method locates a call by the **LINECALLINFO.dwCallID** field. This method may be used by service providers to match up a call appearance on an address to a device call

which has been identified and placed into the **dwCallID** field of the **LINECALLINFO** record. The **dwCallID** field is not used by the class library, and may be used for any purpose by the service provider.

Typically, in the third-party telephony environment, the field is used to tag call objects to switch objects. The application can then determine if two call objects represent the same physical call by comparing the **dwCallID** field.

Return Value

Call which has the specified value in the **LINECALLINFO.dwCallID** field. NULL if no call on this address has that value.

CTSPIAddressInfo::FindCallByHandle

CTSPICallAppearance* FindCallByHandle (HTAPICALL *htCall***) const;**

htCall TAPI opaque call handle to locate.

Remarks

This method locates a call by the TAPI opaque call handle. Each call has a unique handle assigned by TAPI itself to represent the call appearance. It is associated with the call object when the object was created. This method allows the service provider to locate a call appearance using that handle from TAPI.

Return Value

Call which has the specified handle value. NULL if no call on this address matches the specified TAPI call handle.

CTSPIAddressInfo::FindCallByState

CTSPICallAppearance* FindCallByState(DWORD *dwCallState***) const;**

dwCallState **LINECALLSTATE_xxx** to locate.

Remarks

This method locates a call in the specified call state. The first call in the specified state is returned. If multiple calls exist in the same state, then the service provider must enumerate through the calls using **GetCallCount** and **GetCallInfo**.

Return Value

First call found which is currently in the specified state. NULL if no calls on this address are in the specified state.

CTSPIAddressInfo::Forward

virtual LONG Forward(DRV_REQUESTID *dwRequestId***,**
TForwardInfoArray* *parrForwardInfo***,**
DWORD *dwNumRingsNoAnswer***, HTAPICALL** *htConsultCall***,**
LPHDRVCALL *lphdConsultCall***, LPLINECALLPARAMS** *lpCallParams***);**

dwRequestID	Asynchronous request ID associated with this forward request.
lpForwardInfoArray	Array of **TSPIFORWARDINFO** structures (see the section on *Data Structures*).
dwNumRingsNoAnswer	The number of rings to wait before forwarding a call from this address.
htConsultCall	TAPI opaque call handle for the creation of a consultation call as a result of this operation.
lphdConsultCall	TSP call handle to match to the created consultation call created for the forwarding.
lpCallParams	TAPI Call parameters to use for this newly created consultation call.

Remarks

This method is called when an application forwards the address using **lineForward**. The TSP must export the **TSPI_lineForward** function. Some telephony devices transition to the dialtone state when the forwarding function is invoked, for this reason, TAPI sends a call handle to use if this happens. The class library validates that the call is in a valid call state, creates a conference call appearance if the forwarding request generates a consultation call, and finally, submits a **RTForward** asynchronous request into the request queue.

Return Value

TAPI error code or the asynchronous request ID if a **RTForward** request was placed into the request queue.

CTSPIAddressInfo::GatherAgentCapabilities

virtual LONG GatherAgentCapabilities(LPLINEAGENTCAPS *lpAgentCaps***);**

lpAgentCaps	The structure to fill with agent capabilities.

Remarks

This method is called when an application requests the agent capabilities using the **lineGetAgentCaps** function. The class library automatically fills in all the known information about the agent using the stored device and address information.

The service provider can adjust the capabilities returned by using the **GetAgentCaps** method and modifying the returning structure or by calling the supplied member methods which modify the contents.

If the structure is modified through the **GetAgentCaps** method, TAPI is not automatically notified.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GatherAgentStatus

virtual LONG GatherAgentStatus(LPLINEAGENTSTATUS *lpAgentStatus***);**

lpAgentStatus	The structure to fill with the current agent status.

Remarks

This method is called when an application requests the current agent status using the **lineGetAgentStatus** function. The class library automatically fills in all the known information about the agent stored address information.

The service provider can adjust the capabilities returned by using the **GetAgentStatus** method and modifying the returning structure. Some of the information can be modified using other **CTSPIAddressInfo** methods or through the supplied methods which modify various agent status information (**SetAgentState**, **SetAgentFeatures**, etc.)

If the structure is modified through the **GetAgentStatus** method, TAPI is not automatically notified.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GatherCapabilities

virtual LONG GatherCapabilities (DWORD *dwTSPIVersion***,**
DWORD *dwExtVer***, LPLINEADDRESSCAPS** *lpAddressCaps***);**

dwTSPIVersion	The TAPI version which structures should reflect.
dwExtVer	The extension version which structures should reflect.
lpAddressCaps	The structure to fill with address capabilities.

Remarks

This method is called when an application requests the address capabilities using the **lineGetAddressCaps** function. The class library automatically fills in all the known information about the provider using the line, address, and call objects which are associated with the current address object.

The service provider can adjust the capabilities returned by using the **GetAddressCaps** method and modifying the returning structure. This should typically be done when the address is first initialized as capabilities normally do not change during the life of the provider.

If the structure is modified through the **GetAddressCaps** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **LINEADDRESSCAPS** structure.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GatherStatusInformation

virtual LONG GatherStatusInformation (
LPLINEADDRESSSTATUS *lpStatus***);**

lpStatus	The structure to fill with the current address status.

Remarks

This method is called when an application requests the current address status using the **lineGetAddressStatus** function. The class library automatically fills in all the known information about the provider using the line, address, and call objects which are associated with the current address object.

The service provider can adjust the capabilities returned by using the **GetAddressStatus** method and modifying the returning structure. Some of the information can be modified using other **CTSPIAddressInfo** methods. These include **SetNumRingsNoAnswer**, **SetTerminalModes**, **SetCurrentRate**, and **SetAddressFeatures**. Using these methods is recommended as TAPI is notified about the information changing through a **LINEADDRESSSTATUS** callback.

If the structure is modified through the **GetAddressStatus** method, TAPI is not automatically notified.

This method should be overridden if the service provider supports device extensions and wishes to return the information in the **LINEADDRESSSTATUS** structure.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GetAddressCaps

LPLINEADDRESSCAPS GetAddressCaps();
const LPLINEADDRESSCAPS GetAddressCaps();

Remarks

This method returns the structure which maps the capabilities for the address. Any information which is not an offset/size buffer may be modified or read from this structure.

If the structure is modified using this method, TAPI is not automatically notified.

Return Value

Pointer to the structure maintained by the address object.

CTSPIAddressInfo::GetAddressID

DWORD GetAddressID() const;

Remarks

This method returns the address identifier assigned to this address object. The number returned will always be between zero and the total number of addresses on the line owner object.

Return Value

Array index of the address object in question within the line owner array.

CTSPIAddressInfo::GetAddressStatus

LPLINEADDRESSSTATUS GetAddressStatus();
const LPLINEADDRESSSTATUS GetAddressStatus();

Remarks

This method returns the structure which maps the current status for the address. Any information which is not an offset/size buffer may be modified or read from this structure.

If the structure is modified using this method, TAPI is not automatically notified.

Return Value

Pointer to the structure maintained by the address object.

CTSPIAddressInfo::GetAddressType

DWORD GetAddressType() const;

Remarks

This TAPI 3.0 method returns the address type assigned to this address object. This can be any of the **LINEADDRESSTYPE_xxx** constants from TAPI.h and is used by TAPI to indicate the domain type (IP address, phone number, email address, etc.)

Return Value

Assigned address type constant.

CTSPIAddressInfo::GetAgentActivityList

virtual LONG GetAgentActivityList(
LPLINEAGENTACTIVITYLIST *lpActivityList***);**

lpActivityList The structure to fill with the available activities.

Remarks

This method is called when an application requests the list of valid agent activities using the **lineGetAgentActivityList** function. The default implementation returns all the activities defined in the device object (see the reference section on *CTSPIDevice*).

If different addresses support different activities, then this method should be overridden to change the returned values.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GetAgentCaps

const TAgentCaps* GetAgentCaps() const;
TAgentCaps* GetAgentCaps();

Remarks

This method returns the agent capabilities structure used to populate the information for **lineGetAgentCaps**. The fields in the structure may be modified in order to provide new information to TAPI.

Make sure to call **OnAgentCapabilitiesChanged** if any information is modified directly in the structure (rather than using the provided methods).

Return Value

Pointer to the agent capabilities.

CTSPIAddressInfo::GetAgentGroupList

virtual LONG GetAgentGroupList(LPLINEAGENTGROUPLIST *lpGroupList***);**

lpGroupList The structure to fill with the available groups.

Remarks

This method is called when an application requests the list of valid agent groups using the **lineGetAgentGroupList** function. The default implementation returns all the groups defined in the device object (see the reference section on *CTSPIDevice*).

If different addresses support different groups, then this method should be overridden to change the returned values.

Return Value

TAPI Error code or zero if the method was successful.

CTSPIAddressInfo::GetAgentStatus

const TAgentStatus* GetAgentStatus() const;
TAgentStatus* GetAgentStatus();

Remarks

This method returns the agent status structure used to populate the information for **lineGetAgentStatus**. The fields in the structure may be modified in order to provide new information to TAPI.

Make sure to call **OnAgentStatusChanged** if any information is modified directly in the structure (rather than using the provided methods).

Return Value

Pointer to the agent status information.

CTSPIAddressInfo::GetAvailableMediaModes

DWORD GetAvailableMediaModes () const;

Remarks

This method is a method to return the **LINEADDRESSCAPS.dwAvailableMediaModes** value. This was set by the creation of the address object through the media modes passed into the **CTSPILineConnection::CreateAddress** method.

Return Value

The **dwAvailableMediaModes** value in the **LINEADDRESSCAPS** structure.

CTSPIAddressInfo::GetBearerMode

DWORD GetBearerMode() const;

Remarks

This method returns the bearer mode (**LINEBEARERMODE_xxx** value) of the address object. This was set by the creation of the address object through the bearer mode passed into the **CTSPILineConnection::CreateAddress** method.

Return Value

The bearer mode of the address object.

CTSPIAddressInfo::GetCallCount

unsigned int GetCallCount() const;

Remarks

This method is used to return a current count of **CTSPICallAppearance** objects being tracked and owned by this address object. Since the call count changes dynamically while the provider runs this count should not be relied upon for any length of time.

Return Value

The number of call appearance objects owned by this address.

CTSPIAddressInfo::GetCallInfo

CTSPICallAppearance* GetCallInfo(unsigned int *iPos***) const;**

iPos Index of call to retrieve from array.

Remarks

This method returns the call appearance object which is at the specified array position. The position element should be between zero and **GetCallCount**.

Return Value

The call appearance object at the specified position. NULL if no call exists at that position.

CTSPIAddressInfo::GetCallTreatmentName

TString GetCallTreatmentName (DWORD *dwCallTreatment***) const;**

dwCallTreatment Call treatment identifier to return name for.

Remarks

This method returns the call treatment name associated with the specified call treatment identifier. The standard call treatments (**LINECALLTREATMENT_xxx**) or service-provider specific treatments can be added to the address using **AddCallTreatment**.

Return Value

The name of the specified call treatment.

CTSPIAddressInfo::GetCompletionMessage

LPCTSTR GetCompletionMessage (unsigned int *iPos*) const;

iPos Array position of the completion message.

Remarks

This method returns the completion message associated with the specified array position index. Completion messages may be added to the address using **AddCompletionMessage**.

Return Value

The description of the specified completion message.

CTSPIAddressInfo::GetCompletionMessageCount

unsigned int GetCompletionMessageCount() const;

Remarks

This method returns the count of completion messages associated with the address. Completion messages may be added to the address using **AddCompletionMessage**.

Return Value

The total number of completion messages on the address. Zero if no completion messages exist.

CTSPIAddressInfo::GetCurrentAgentGroup

const TAgentGroup* GetCurrentAgentGroup(unsigned int *iPos*) const;

iPos Index of the group to retrieve (0 – **GetCurrentAgentGroupCount**).

Remarks

This method returns a specific agent group structure that this address is logged into. The groups are added using **AddAgentGroup** or by completing a **RTSetAgentGroup** request successfully.

Return Value

Pointer to an agent group structure.

CTSPIAddressInfo::GetCurrentAgentGroupCount

unsigned int GetCurrentAgentGroupCount() const;

Remarks

This method returns the number of groups this address is currently logged into. The actual group information can be retrieved using the **GetCurrentAgentGroup** method.

Return Value

Numeric number of groups the address is logged into, zero if none.

CTSPIAddressInfo::GetCurrentAgentState

DWORD GetCurrentAgentState() const;

Remarks

This method returns the current agent state according the **TAgentStatus** structure stored in the address.

Return Value

The **LINEAGENTSTATE_xxx** constant representing the current agent state.

CTSPIAddressInfo::GetCurrentRate

DWORD GetCurrentRate() const;

Remarks

This method returns the current data rate associated with the address. It is initialized during the address initialization to the minimum data rate and may be adjusted using **CTSPIAddressInfo::SetCurrentRate**.

Return Value

The current data rate associated with the address object.

CTSPIAddressInfo::GetDeviceClass

DEVICECLASSINFO* GetDeviceClass(LPCTSTR *pszClass***);**

pszClass Device class to search for (e.g. "tapi/line", etc.)

Remarks

This method returns the device class structure associated with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEDEVCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

DEVICECLASSINFO structure which represents the specified text name. NULL if no association has been performed.

CTSPIAddressInfo::GetDialableAddress

LPCTSTR GetDialableAddress() const;

Remarks

This method returns the dialable phone number of the address object. This *dialable address* is associated with the address object when it was created through the **CTSPILineConnection::CreateAddress** method.

Return Value

The dialable address of the address object.

CTSPIAddressInfo::GetID

virtual LONG GetID (TString& *strDevClass***, LPVARSTRING** *lpDeviceID***, HANDLE** *hTargetProcess***);**

strDeviceClass	Device class which is being asked for.
lpDeviceID	Returning device information.
hTargetProcess	Process requesting data.

Remarks

This method is called in response to an application calling **lineGetID**. It is used to return device information to the application based on a string device class key. TSP++ implements this method internally and returns any device information which was added using **CTSPIAddressInfo::AddDeviceClass**.

Any handle which was given to the **AddDeviceClass** method is automatically duplicated in the given process for TAPI 2.x using the **DuplicateHandle** Win32 function.

Return Value

TAPI error code or zero if the method was successful.

CTSPIAddressInfo::GetLineOwner

CTSPILineConnection* GetLineOwner() const;

Remarks

This method returns the owning line object for this address.

Return Value

Pointer to the owner **CTSPILineConnection** object of this address.

CTSPIAddressInfo::GetName

LPCTSTR GetName() const;

Remarks

This method returns the name of the address object. The name was established when the address object was created through the **CTSPILineConnection::CreateAddress** method. It may be changed using the **CTSPIAddressInfo::SetName** method.

This name is returned in the **LINEADDRESSCAPS** structure and is used by many applications to represent the address on dialogs.

Return Value

Pointer to buffer with name of the address object.

CTSPIAddressInfo::GetTerminalInformation

Protected

DWORD GetTerminalInformation (int *iTerminalID***) const;**

iTerminal	Terminal identifier to retrieve information for.

Remarks

This method returns the terminal information which applies to the address. The terminal information is automatically copied from the line owner object when the address is created and is kept in synch with any changes to the line or calls below it.

Return Value

Pointer to buffer with name of the address object.

CTSPIAddressInfo::Init

Protected

virtual void Init (CTSPILineConnection* *pLine***, DWORD** *dwAddressID***,**
LPCTSTR *lpszAddress***, LPCTSTR** *lpszName***, bool** *fIncoming***,**
bool *fOutgoing***, DWORD** *dwAvailMediaModes***,**
DWORD *dwlBearerMode***, DWORD** *dwMinRate***, DWORD** *dwMaxRate***,**
DWORD *dwMaxNumActiveCalls***, DWORD** *dwMaxNumOnHoldCalls***,**
DWORD *dwMaxNumOnHoldPendCalls***, DWORD** *dwMaxNumConference***,**
DWORD *dwMaxNumTransConf***, DWORD** *dwAddressSharing***,**
DWORD *dwAddressType***);**

pLine	Line owner object for this address.
dwAddressID	Address index for this object
lpszAddress	Dialable phone number of the address.
lpszName	ASCII name reported back in **ADDRESSCAPS**.
fIncoming	**TRUE** if incoming calls are allowed on this address.
fOutgoing	**TRUE** if outgoing calls are allowed on this address.
dwAvailMediaModes	Available media modes on this address.
dwBearerMode	Single **LINEBEARERMODE_xxx** flag.
dwMinRate	Minimum data rate reported in **ADDRESSCAPS**.
dwMaxRate	Maximum data rate reported in **ADDRESSCAPS**.
d*wMaxNumActiveCalls*	Max number of calls in a **Connected** state.
d*wMaxNumOnHoldCalls*	Max number of calls in a **Hold** state.
dwMaxNumOnHoldPendCalls	Max number of calls waiting for **Transfer**/**Conference**.
d*wMaxNumConference*	Max number of calls conferenced together.
d*wMaxNumTransConf*	Max number of calls conferenced from a transfer event.
dwAddressSharing	Single **LINEADDRESSSHARING_xxx** mode.
dwAddressType	**(TAPI3)** Single **LINEADDRESSTYPE_xxx** type

Remarks

This method is used to initialize an address object. It is called directly after the constructor of the **CTSPIAddressInfo** object in response to a **CTSPILineConnection::CreateAddress** call. If the service provider overrides this method, it *must* call the base class implementation.

CTSPIAddressInfo::MoveCall

CTSPICallAppearance* MoveCall(CTSPICallAppearance* *pCall*,
DWORD *dwReason*=**0, DWORD** *dwState*=**0, DWORD** *dwStateMode*=**0**
DWORD *dwTimeout* **= 0);**

pCall	Call to move to this address.
dwReason	**LINECALLREASON_xxx** to report with moved call.
dwState	State of moved call on our address.
dwStateMode	Mode (for callstate) of moved call on our address.
dwTimeout	Time (in mSecs) to wait for a call to be created on the destination line. Use this if the hardware sends a create event before transfer events.

Remarks

This method can be used to move a call from one line to another. The passed call is copied into a new call object created on the address, the reason and state are set into the new call and the original call is changed to the **LINECALLSTATE_IDLE** state.

If the reason code is not passed, then the call inheirits the original reason code from the passed call. The same is true of the callstate.

This method is designed to be used with line-to-line transfers or routes within a service provider.

Return Value

The created call appearance on the address.

CTSPIAddressInfo::NotifyInUseZero

virtual bool NotifyInUseZero();

Remarks

This method is called by the address object when any calls change state on the address (during the **OnCallStateChanged** method). It determines if the address is still in-use and reports the **LINEADDRESSSTATE_INUSE** constants back to TAPI. It may be overridden to supply other behavior when the address use count drops.

Return Value

Boolean value indicating whether the address has non-idle calls present.

CTSPIAddressInfo::OnAddressCapabilitiesChanged

Protected

virtual void OnAddressCapabiltiesChanged();

Remarks

This method is called when any information within the **LINEADDRESSCAPS** structure changes. It is called by the class library when any of the **SetXXX** methods are called which change information in the **LINEADDRESSCAPS** structure. It should be called by the service provider if any of the data within the structure is changed through the **GetAddressCaps()** method.

The default behavior is to notify TAPI through a **LINEADDRESSSTATE_CAPSCHANGE** event.

CTSPIAddressInfo::OnAddressFeaturesChanged

Protected

virtual DWORD OnAddressFeaturesChanged (DWORD dwFeatures);

dwFeatures	New features bitmask for the address (**LINEADDRFEATURE_x**).

Remarks

This method is called when the **LINEADDRESSCAPS.dwAddressFeatures** bits are about to be changed by the class library. It gives the derived provider an opportunity to adjust the capabilities before TAPI is notified.

Return Value

Adjusted feature flags for the address. The default return value is the passed in **dwFeatures** bitmask.

CTSPIAddressInfo::OnAddressStateChange

Protected

virtual void OnAddressStateChange (DWORD *dwAddressState***);**

dwAddressState	Address state change to send to TAPI.

Remarks

This method is used by the class library to notify TAPI about events occurring on the address. It checks to see if the address state notification type is being monitored by TAPI and then sends a **LINE_ADDRESSSTATE** event notification if it is.

CTSPIAddressInfo::OnAgentCapabilitiesChanged

virtual void OnAgentCapabiltiesChanged();

Remarks

This method is used by the class library to notify TAPI about changes in the agent capabilities structure.

CTSPIAddressInfo::OnAgentStatusChanged

virtual void OnAgentStatusChanged(DWORD *dwState*, **DWORD** *dwP2*=**0L);**

dwState	New agent state
dwP2	Optional parameter for notification.

Remarks

This method is used by the class library to notify TAPI about changes in the agent status structure.

CTSPIAddressInfo::OnCallFeaturesChanged

Protected

virtual DWORD OnCallFeaturesChanged(CTSPICallAppearance* *pCall*,
DWORD *dwCallFeatures*);

pCall	Call appearance on this address which has changed.
dwCallFeatures	New call feature bitmask for the call (**LINECALLFEATURE_x**).

Remarks

This method is called by the child call appearance object when the feature list for the call is about to be changed by the class library. It gives the address object an opportunity to adjust its own feature list based on what is now available on the call.

The default class library behavior is to call the **CTSPILineConnection::OnCallFeaturesChanged** method.

Return Value

Adjusted feature flags for the call. The default return value is the passed in **dwCallFeatures** bitmask.

CTSPIAddressInfo::OnCallStateChanged

virtual void OnCallStateChange (CTSPICallAppearance* *pCall*,
DWORD *dwState*, **DWORD** *dwOldState*);

pCall	Call appearance on this address which has changed.
dwState	New call state for the call.
dwOldState	Previous call state for the call.

Remarks

This method is called by the child call appearance object when the call state of the call has changed. It gives the address object an opportunity to adjust its feature list based on what is now available on the call. TAPI has already been notified when this is called.

The default class library behavior is to call the **CTSPILineConnection::OnCallStateChanged** method.

CTSPIAddressInfo::OnCreateCall

Protected

virtual void OnCreateCall (CTSPICallAppearance* *pCall***);**

pCall	New Call appearance on this address.

Remarks

This method is called when any call appearance (conference, consultant or normal) is created on the address.

The default class library behavior is to ignore the event.

CTSPIAddressInfo::OnPreCallStateChanged

virtual void OnPreCallStateChange (CTSPICallAppearance* *pCall***,**
DWORD *dwState***, DWORD** *dwOldState***);**

pCall	Call appearance on this address which has changed.
dwState	New call state for the call.
dwOldState	Previous call state for the call.

Remarks

This method is called by the child call appearance object when the call state of the call is about to be changed. It gives the address object an opportunity to adjust its feature list based on what is now available on the call. This is called *before* TAPI is notified.

The default class library behavior is to adjust the call counters for the address and then call the **CTSPILineConnection::OnPreCallStateChanged** method.

CTSPIAddressInfo::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq***, LONG** *lResult***);**

pReql	Request which has completed
lResult	Final return code for the request.

Remarks

This method is called when a request is completed on the owner address/line. It gives the address object an opportunity to cleanup any data or information associated with the request.

The default implementation of the class library manages several requests:

RTSetTerminal

If the request completes successfully, the terminal information is updated in the address object using the **CTSPIAddressInfo::SetTerminalModes** method.

RTForward

If the request completes successfully, the forwarding information is updated on the address.

CTSPIAddressInfo::OnTerminalCountChanged

Protected

virtual void OnTerminalCountChanged (bool *fAdded***, int** *iPos***,**
DWORD *dwMode* **= 0L);**

fAdded	Whether the terminal mode was ADDED or REMOVED.
iPos	Position of the affected terminal.
dwMode	Terminal mode which was added or removed.

Remarks

This method is used by the class library to synchronize the terminal modes between the line and address objects. When the terminal information is adjusted through TAPI using the **lineSetTerminal** function, this method adjusts each of the addresses on the line to match the new terminal modes. The terminal mode is either *added* to the existing terminal, or *removed* depending on the first parameter.

CTSPIAddressInfo::Pickup

virtual LONG Pickup (DRV_REQUESTID *dwRequestID***, HTAPICALL** *htCall***,**
LPHDRVCALL *lphdCall***, TSPILINEPICKUP*** *lpPickup***);**

dwRequestID	Asynchronous request ID for this request.
htCall	New call appearance handle from TAPI.
lphdCall	Returning call handle from TSP.
lpPickup	Data structure for request.

Remarks

This method is called in response to an application calling **linePickup**. It is used to picks up a call alerting at the specified destination address and returns a call handle for the picked-up call. The class library will validate the basics of the request and create a **RTPickup** request packet and insert it into the line owner queue.

Return Value

TAPI error code or asynchronous request ID if the **RTPickup** packet was added to the queue.

CTSPIAddressInfo::read

virtual std::istream& read(std::istream& *istm***);**

istm	input iostream to read information from.

Remarks

This method is called during initialization if it is determined that the line/address object information is contained within the registry. This is only used if the SPLUI library stored object information in the registry.

It may be overridden to read additional information from the stream (other than the information placed there by TSP++).

Note: you *must* retrieve information in the same order as it was stored in the SPLUI user-interface DLL implementation for your provider! If the TSP locks up on loading then check the serialization to ensure you are not reading past the iostream.

CTSPIAddressInfo::RecalcAddrFeatures

virtual void RecalcAddrFeatures();

Remarks

This method is called when the address status is changed to re-determine the address's feature set based on the status and call information.

It may be overridden to catch all changes to the address state.

CTSPIAddressInfo::RemoveAllAgentGroups

void RemoveAllAgentGroups();

Remarks

This method removes all the agent groups associated with this address.

This method *does not* notify TAPI that the agent status has changed – after calling this method, call the **OnAgentStatusChanged** method.

CTSPIAddressInfo::RemoveCallAppearance

void RemoveCallAppearance(CTSPICallAppearance* *pCall***);**

pCall	Call appearance to remove from the array.

Remarks

This method is used by the class library to remove and delete calls from the address call array. The method deletes the call object passed so once this returns, the specified call object memory address will be invalid. It is typically called when TAPI deletes a call appearance through **TSPI_lineCloseCall**, or when the service provider fails a request packet containing a consultation call.

CTSPIAddressInfo::RemoveCallTreatment

void RemoveCallTreatment (DWORD *dwCallTreatment***);**

dwCallTreatment	Call treatment index to remove.

Remarks

This method may be used remove any added call treatment entries. Call treatment entries are added using **CTSPIAddressInfo::AddCallTreatment**. The current call treatment indicates the sounds a party on a call that is unanswered or on hold hears.

CTSPIAddressInfo::RemoveDeviceClass

bool CTSPIAddressInfo::RemoveDeviceClass (LPCTSTR *pszClass***);**

pszClass Device class key to remove.

Remarks

This method removes the specified device class information from this object. It will no longer be reported as available to TAPI.

Return Value

TRUE if device class information was located and removed.

CTSPIAddressInfo::SetAddressFeatures

void SetAddressFeatures(DWORD *dwFeatures***);**

dwFeatures Address features which are now available (**LINEADDRFEATURE_xx**).

Remarks

This method sets the current features available on the address It does *not* invoke the **CTSPIAddressInfo::OnAddressFeaturesChanged** method.

CTSPIAddressInfo::SetAgentActivity

void SetAgentActivity(DWORD dwActivity);

dwActivity New activity for the agent.

Remarks

This method changes the current agent's activity on the address. The activity should be available within the owner device list (see the reference section on *CTSPIDevice*).

This method is automatically called when a **lineSetAgentActivity** request is completed successfully.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::SetAgentActivity

virtual LONG SetAgentActivity(DRV_REQUESTID *dwRequestID***,**
DWORD *dwActivity***);**

dwRequestID TAPI request id for asynchronous operation
dwActivity Requested current activity for the agent.

Remarks

This method is called when TAPI invokes the **lineSetAgentActivity** function. It validates the parameters and then generates a **RTSetAgentActivity** request and queues it to the line object for handling.

Return Value

TAPI result code

CTSPIAddressInfo::SetAgentFeatures

void SetAgentFeatures(DWORD *dwFeatures***);**

dwFeatures	New bitmask of **LINEAGENTFEATURE_xxx** which are available currently.

Remarks

This method changes available agent features for the address.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::SetAgentGroup

void SetAgentGroup(TAgentGroupArray* *parrGroups***);**

parrGroups	Array of **TAgentGroup** structures to log the address into.

Remarks

This method changes the current agent's group assignments on this address. If the array is NULL or contains no data the agent is automatically logged out of the address.

This method is automatically called when a **lineSetAgentGroup** method is completed successfully.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::SetAgentGroup

virtual LONG SetAgentGroup(DRV_REQUESTID *dwRequestID***,**
LPLINEAGENTGROUPLIST *lpGroupList***);**

dwRequestID	TAPI request id for asynchronous operation
lpGroupList	TAPI **LINEAGENTGROUPLIST** to log this address into.

Remarks

This method is called when TAPI invokes the **lineSetAgentGroup** function. It validates the parameters and then generates a **RTSetAgentGroup** request and queues it to the line object for handling.

Return Value

TAPI result code

CTSPIAddressInfo::SetAgentState

void SetAgentState(DWORD *dwState*, **DWORD** *dwNextState*);

dwState	New state for the agent.
dwNextState	Next state for the agent.

Remarks

This method changes the current agent's state on the address. If the state is set to **LINEAGENTSTATE_LOGGEDOFF** then all the agent groups are automatically removed from the group array.

The *dwNextState* method can be 0 if the next state is not known.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::SetAgentState

virtual LONG SetAgentState(DRV_REQUESTID *dwRequestID*,
DWORD *dwState*, **DWORD** *dwNextState*);

dwRequestID	TAPI request id for asynchronous operation
dwState	New requested state for the agent
dwNextState	Requested *next* state for the agent.

Remarks

This method is called when TAPI invokes the **lineSetAgentState** function. It validates the parameters and then generates a **RTSetAgentState** request and queues it to the line object for handling.

Return Value

TAPI result code

CTSPIAddressInfo::SetCurrentRate

void SetCurrentRate (DWORD *dwRate*);

dwRate	Current data rate which is being used on the address.

Remarks

This method sets the current data rate being used on the address. This should generally be called for a modem-style device or ISDN address where a known data rate is set on the device. This will change the initial data rate of any call created on the address after this completes.

CTSPIAddressInfo::SetDialableAddress

void SetDialableAddress(LPCTSTR *pszAddress*);

pszAddress	New dialable address for this object.

Remarks

This method changes the dialable address of the address object. This typically will not be used as the dialable address normally doesn't change during the life of a provider. If the extension *can* change, this method can be used to adjust the **LINEADDRESSCAPS.dwDialableAddr** fields. TAPI is notified that the **LINEADDRESSCAPS** structure has changed.

CTSPIAddressInfo::SetMediaControl

virtual LONG SetMediaControl (TSPIMEDIACONTROL* *lpMediaControl***);**

lpMediaControl Data structure for request.

Remarks

This method is called in response to an application calling **lineSetMediaControl**. The **TSPI_lineSetMediaControl** must be exported by the service provider to support this method. It enables and disables control actions on the media stream associated with the address. Media control actions can be triggered by the detection of specified digits, media modes, custom tones, and call states. The class library will save off the new media control packet and then enumerate through each of the call appearances on the address and notify them that the media control information has changed using the **CTSPICallAppearance::SetMediaControl** method.

Return Value

TAPI error code or zero if the method was successful.

CTSPIAddressInfo::SetName

void SetName (LPCTSTR *pszName***);**

pszName ASCII name for the address.

Remarks

This method changes the name reported in the caller id field for any outgoing calls.

CTSPIAddressInfo::SetNumRingsNoAnswer

void SetNumRingsNoAnswer (DWORD *dwNumRings***);**

dwNumRings Number of rings before answering.

Remarks

This method may be used by the service provider to change the **LINEADDRESSSTATUS.dwNumRingsNoAnswer** field which is used for forwarding information. It notifies TAPI that the field has changed using a **LINEADDRESSSTATE_FORWARD** event notification.

CTSPIAddressInfo::SetStatusMessages

virtual void SetStatusMessages(DWORD *dwStates***);**

dwStates	New status messages to send to TAPI.

Remarks

This method is called in response to an application calling **lineSetStatusMessages**. It changes the notifications which the service provider forwards onto TAPI. It is completely implemented within the class library and only requires that the service provider exports **TSPI_lineSetStatusMessages**.

Return Value

TAPI error code or zero if the method was successful.

CTSPIAddressInfo::SetTerminalModes

void SetTerminalModes (int iTerminalID, DWORD *dwTerminalModes***,**
bool *fRouteToTerminal***);**

iTerminalID	Terminal identifier to adjust (0 to **GetTerminalCount**).
dwTerminalModes	Terminal mode(s) (**LINETERMMODE_xxx**) to adjust.
f*RouteToTerminal*	Whether the terminal is added or removed from modes.

Remarks

This is the method which is called when a **RTSetTerminal** is completed by the service provider. This stores or removes the specified terminal from the terminal modes given, and then forces it to happen for any existing calls on the address by routing the notification through the **CTSPICallAppearance::SetTerminalModes**.

TAPI is notified about the change through a **LINEDEVSTATE_TERMINALS** event.

CTSPIAddressInfo::SetupConference

virtual LONG SetupConference (DRV_REQUESTID *dwRequestID***,**
CTSPICallAppearance* *pCall***, HTAPICALL** *htConfCall***,**
LPHDRVCALL *lphdConfCall***, HTAPICALL** *htConsultCall***,**
LPHDRVCALL *lphdConsultCall***, DWORD** *dwNumParties***,**
LPLINECALLPARAMS *lpCallParams***);**

dwRequestID	Asynchronous request ID for this request.
pCall	Existing call to start the conference with.
htConfCall	New TAPI handle for created conference call.
lphdConfCall	Returning call handle for created conference call.
htConsultCall	New TAPI handle for consultation call.
lphdConsultCall	Returning call handle for created consultation call.
dwNumParties	The number of parties that the application intends to add to the conference.

lpCallParams	**LINECALLPARAMS** for the newly created consultation call.

Remarks

This method is called in response to an application calling **lineSetupConference**. The **TSPI_lineSetupConference** function must be exported from the provider. The address object validates the call which is starting the conference, validates the passed parameters, and creates the new conference call and possibly, the consultation call. It then attaches the consultation call to the conference using **CTSPICallAppearance::AttachCall** and submits a **RTSetupConference** request into the request queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPIAddressInfo::SetupTransfer

virtual LONG SetupTransfer(DRV_REQUESTID *dwRequestID***,**
CTSPICallAppearance* *pCall***, HTAPICALL** *htConsultCall***,**
LPHDRVCALL *lphdConsultCall***, LPLINECALLPARAMS** *lpCallParams***);**

dwRequestID	Asynchronous request ID for this request.
pCall	Existing call to transfer.
htConsultCall	New TAPI handle for consultation call.
lphdConsultCall	Returning call handle for created consultation call.
lpCallParams	**LINECALLPARAMS** for the newly created consultation call.

Remarks

This method is called in response to an application calling **lineSetupTransfer**. The **TSPI_lineSetupTransfer** function must be exported from the provider. This method is used for managing a supervised transfer where an intermediate call is placed by the application. The class library validates the passed parameters, creates the consultation call, and then submits a **RTSetupTransfer** request into the queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPIAddressInfo::SetValidAgentStates

void SetValidAgentStates(DWORD *dwStates***);**

dwStates	New states the agent may use.

Remarks

This method changes the current agent's available states which can be used with **lineSetAgentState**.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::SetValidNextAgentStates

void SetValidNextAgentStates(DWORD *dwStates***);**

dwStates	New states the agent may use.

Remarks

This method changes the current agent's available *next* states which can be used with **lineSetAgentState**.

This method changes the internal state and notifies TAPI that the agent status has changed.

CTSPIAddressInfo::Upark

virtual LONG Unpark (DRV_REQUESTID *dwRequestID***, HTAPICALL** *htCall***, LPHDRVCALL** *lphdCall***, TDialStringArray*** *parrAddresses***);**

dwRequestID	Asynchronous request ID for this request.
htCall	New TAPI handle for unparked call.
lphdCall	Returning call handle for unparked call.
parrAddresses	Addresses to unpark call from.

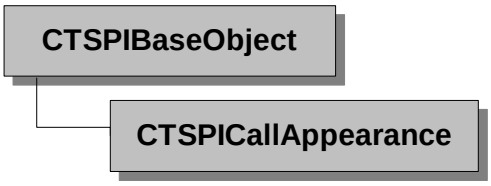
Remarks

This method is called in response to an application calling **lineUnpark**. The **TSPI_lineUnpark** function must be exported from the provider. This method is used to retrieve a parked call at the address specified in the address array. The class library validates that another call can be created, creates a call appearance to model the unparked call, and then calls the **CTSPICallAppearance::Unpark** method to retrieve the call.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance



Each **CTSPICallAppearance** object represents a call on an address. The calls are maintained in a list by the **CTSPIAddressInfo** object, and are dynamically created and destroyed as needed. The call appearance object stores the capabilities and status for the call. Each request from the TAPI server which is targeted for an address or a call will eventually get to one of these objects. The call appearance object generally represents an end-point to a conversation being maintained on the address. It doesn't need to have a direct connection to a physical call on the telephone network although it can, and generally will. In some situations, multiple call appearances may be present on a single address, although in general, only one of these calls will be *active* at any given point. The actual connection between the call appearance and the physical hardware is left up to the service provider to establish and determine (the field **dwCallID** in the **LINECALLINFO** structure may be used to store any information required).

Call Appearances

Unlike lines and addresses, call appearances are dynamic. A call appearance represents a connection between two or more addresses. The originating address is the caller. The destination address identifies the remote endpoint or station which the originator dialed (the called). Zero, one, or more calls can exist on a single address at any given time. A good example of multiple calls on a single address is call waiting: during a conversation with one party, an address with call waiting is notified (through some mechanism) that another party is attempting to call. The phone is then "flashed" to answer the second call which places the first party on hold. The person can then toggle between the two by flashing and talk to either party. In this example, the person has two calls using a single address. Since the telephone handset can only talk to one remote party at a time, only one call appearance may be active at any given point, the other being placed on hold at the switch.

Call Handles

TAPI identifies a specific call by a *call handle*. One call handle exists for each call owned or monitored by the TAPI server. Each call created will get assigned a TAPI handle, and when new calls are received on the device, the service provider should tell the address to create a new call appearance with the **CreateCallAppearance** method of the **CTSPIAddressInfo** object, passing a NULL in for the TAPI handle. This will indicate to ask TAPI for a new call handle, and the resulting call appearance object will be connected to it. When a call is deleted through a **lineDeallocateCall**, the address object will automatically delete the call appearance and remove it from the call array.

Call States

Each call created has a *call state*. This state defines to TAPI applications what is happening on the call and which functions are currently available. Initially the call will be in the **Unknown** state. As different events are performed on the call, it will transition through various call states, eventually ending up in the **Idle** state. At this point, the call is considered "dead" until it is de-allocated by TAPI through a **lineDeallocateCall** request. The call appearance should *never* transition out of an **Idle** state. As the state of the call changes, TAPI is automatically notified by the class library, and the other objects (line and address) are also told about the call and adjust the available features for their own object based on what state the call is in. The state of the call is generally not changed by the class library, it is left up to the derived service provider class to adjust the call state as the device notifies it about changes on the call.

Call states can change as a result of the TAPI server performing some action with the call, an unsolicited event caused by the switch or telephone network, or the user or remote party pressing buttons on the phone. Also, different call states can indicate that connections exist to different parts of the switch. For example, a *dial tone* is a particular state of a switch that means the computer is ready to receive digits to dial.

Whenever a call changes state, the TAPI server is notified and will report the new state to the application in a callback. This call-state notification tells the application what the call's new state is, instead of reporting the occurrence of specific events and assuming that the application will be able to deduce the transitions between two states.

Some of the call states and events defined by TAPI are exclusive to inbound or outbound call processing, while others occur in both cases. Several of these call states provide additional information that can be used by the application. For example, the *busy* state signifies that a call cannot be completed because a resource between the originator and the destination is unavailable, as when an intermediate switch has reached its capacity and cannot handle an additional call. Information supplied with the *busy* state includes *station busy* or *trunk busy*. Station busy means that the destination's station is busy (the phone is off-hook), while trunk busy means that a circuit in the switch or network is busy.

Bearer and Media Modes

The *bearer mode* of a call corresponds to the quality of service requested from the network for establishing a call. The *media mode* of a call describes the type of information that is exchanged over a specific call of a given bearer mode. As an example, the analog telephone network provides only 3.1 kHz voice-grade quality of service - this is its bearer mode. However, a call with this bearer mode can support a variety of different media modes such as voice, fax, or data modem. In other words, media modes require certain bearer modes. The bearer mode of a call is specified when the call is set up, or is provided when the call is offered. With line devices able to represent channel pools, it is possible for a service provider to allow calls to be established with wider bandwidth, this could be possible with ISDN for instance by tying the "A" and "B" channels together.

New Call Media mode identification

When the service provider detects a new call on the network, it generally will offer the call to the TAPI server. This occurs by creating a new call appearance on the address -

using the **CTSPIAddressInfo::CreateCallAppearance** and passing a NULL for the call handle. Then the call state should be set to **LINECALLSTATE_OFFERING** to indicate that the call is requesting pickup by the station. When the call is offered, a suggested media mode must be passed to indicate which application should receive ownership of the call. No call may exist in TAPI without an owner, if a call is handed to TAPI in the **Connected** state, and no owner is found, it will be dropped. TAPI will select the media modes it is watching for by calling the **lineSetDefaultMediaDetection** method. This in turn can be queried by the service provider by the **CTSPILineConnection:: GetDefaultMediaDetection** method. Any call offered to TAPI which has a media mode not in the selected list will be ignored or dropped. It is suggested, though not necessary, that the service provider keep track of the call, instead of offering it to TAPI, and if the **lineSetMediaDetection** is called again with the selected media mode present, *then* offer the call to TAPI if it is still available. This cuts down on the process that the TAPI server must perform to determine ownership.

When the service provider detects a new call, it may be able to single out a media mode, or it may only be able to narrow down the possibilities to a certain few. These first media mode settings are called *initial media modes*, and are passed to TAPI using the **SetCallState** method when the new call is set to **LINECALLSTATE_OFFERING**. If the media mode can be positively identified, only one flag will be set in the initial media mode - that of the call being offered. Otherwise, the multiple flags will be set, and the **LINEMEDIAMODE_UNKNOWN** bit will also be set indicating that the media mode has not been completely determined.

The service provider can use several methods to narrow down the media modes of an offering call. Many of them revolve around configuration issues. For instance, the service provider could be configured to:

- Work with only a single media mode, or certain media modes.
- Associate particular called addresses with particular media modes. This can be accomplished using **Direct Inward Dialing** (DID).
- Associate particular caller addresses with particular media modes. This can be accomplished using Caller ID.
- Identify the ring pattern of the incoming call and match it to a predetermined pattern (of several possibilities) that is reserved for a particular media mode. For example, if the incoming call is using ring pattern 2, the service provider could be configured to recognize it to be a fax call.

Also, depending on the intelligence of the network, the service provider might be able to analyze the call's protocol frames to determine the media mode. Or, the provider might automatically answer the call and perform probing on the line to determine the media mode. In this scenario, the provider would pass the call to TAPI in the **Connected** state.

Once the call is answered by an owner application, the TAPI application is responsible for media probing. If during this process, the service provider detects a media change on the line, it should use the **OnDetectedNewMediaModes** method, adding whatever flags are necessary. For example, an incoming call comes in on a line, and the service provider determines that the media could be **InteractiveVoice**, **Fax**, or **DataModem**. In this case, it will create a new call appearance (through the **CreateCallAppearance** method of the **CTSPIAddressInfo** which the call is being offered on), and send an initial callstate change of **Offering** with the initial media modes being **InteractiveVoice**, **Fax**, **DataModem**, and **Unknown** since it cannot positively identify the media mode. The TAPI server will then begin a laborious process of finding a call owner for this call based on a priority of applications and media-type handoffs (for more information on this, see

the *Windows SDK Reference: TAPI*). Since **InteractiveVoice** is the highest priority media mode, it will be tested first. If the application conclusively determines that there is a human caller at the destination, it will call **lineSetMediaMode** to change the media mode field of the **LINECALLINFO** record. But, if during the probing being performed by the application, the media state suddenly becomes known to the service provider, then the provider will call the **OnDetectedNewMediaModes** and send TAPI any monitoring information it needs. As an example, the application might play an outgoing "leave a message" voice message while the incoming call starts sending a fax calling tone. By calling the **OnDetectedNewMediaModes**, the media monitoring will be checked and potentially inform TAPI of the new incoming media mode. Note that the **LINECALLINFO** record is *not* changed in this case. Once the initial media mode is determined, the service provider should never change this field unless requested by the TAPI server.

Shadow Calls and Call Hubs

When modeling a PBX or ACD switch, the provider sometimes can "see" both sides of a conversation. In these cases, the service provider is presented with a single call-id but multiple call objects (since the call appears on more than one line device). This causes the creation of a *shadow* relationship (Microsoft documentation refers to this as "half-calls"). A shadow call is the call object that represents the other side of an active conversation. The call appearances are related because they represent a single channel where data is flowing, so if one changes state, many times the other will as well. For example, when one side disconnects, the other side must also disconnect (although it may not go idle until the application actually drops the call – in the same way that you could continue to keep the handset off the hookswitch even though the connection is gone).

Each unique call-id is associated with a *call hub* object in the TSP++ library. This hub is a holding place for all the calls that share that call-id. Normally this will be one or two calls, but on some switches, conference calls share a single call-id and transfers merge multiple calls into a single call-id. In these situations, three or more calls might be part of the call hub for a given call-id. The call hub relationship is managed completely within TSP++ and is updated as each call changes its call-id or moves across lines.

The shadow call can be retrieved using the **GetShadowCall** method, and the call hub can be retrieved using **GetCallHub**. Note that when more than two calls share a call-id, the shadow call is ambiguous and the method will return NULL – in these situations, the derived provider must determine which call is really the other side from the call hub information.

The call hub will always exist as long as the call-id for a call is non-zero.

Call Appearance Initialization

The call appearance objects are dynamically created by the **CTSPIAddressInfo** object as needed. They can be created due to a variety of reasons, for instance, a new call being placed on the address (**lineMakeCall**) or a new called being offered on the address. Also, several of the TAPI functions may require the usage of a *consultation call* in order to complete. As part of the initialization of a call appearance, the default settings for the **LINECALLINFO** and **LINECALLSTATUS** records are filled out.

Call Information

The current information for a call is stored in the **CTSPICallAppearance** object in a **LINECALLINFO** structure. TAPI will ask for this information periodically through the **lineGetCallInfo** method of the service provider. Most of the fields are handled automatically by the base class, and some are supplied to the **CreateCallAppearance** method when the call is created on the address. It is the responsibility of the service provider in most cases to supply the call reason. Since many of the reasons are for incoming calls, the library cannot determine this information and therefore the service provider should supply that information when the call is created on the address.

The pieces of information not supported by the base class include comments, user-to user information, high and low level compatibility information, charging information, and device-specific extensions. All of these use the variable portions of the data structure, and therefore to supply them, the service provider will have to either override **lineGetCallInfo** in the **CServiceProvider** class, or derive a new call appearance and override **GatherCallInformation**.

Call Status

The current call status is also stored in the **CTSPICallAppearance** object. It is stored in a **LINECALLSTATUS** structure which is embedded in the class. All of the required information within the structure is maintained completely within the library. The only fields not completed in the library are the device-specific extension fields. Since this field is variable, to be supported in the service provider will require either an override of the **lineGetCallStatus** method of the **CServiceProvider** class, or a derivation of a new call appearance and overriding the **GatherCallStatus** method.

Device Specific Extensions

If the **LINECALLINFO** or **LINECALLSTATUS** records can supports device-specific information, then these must be added either by overriding the **CServiceProvider** method **lineGetCallInfo/lineGetCallStatus**, or by overriding the **CTSPICallAppearance** methods **GatherCallInformation/GatherStatusInformation**.

> **Note:** Do not add the device specific information into the static LINECALLINFO or LINECALLSTATUS structures, there is no extra allocated space!

Request completions

When a request completes on the line, the **OnRequestComplete** method is called. The default line connection object takes the following actions when processing requests:

RTSetTerminal If the request completes successfully, the call appearance will alter the internal terminal map and notify TAPI that the terminal information has changed in the **LINECALLINFO** record.

RTSetCallParams If the request succeeds, the bearer mode and dialing parameters of our call information record will be altered to reflect the new changes.

RTSecureCall If the request succeeds, the call information record will be altered to reflect that the call is now *secure*.
RTGenerateDigits If the request succeeds, TAPI will be notified about the digit generation completion through a **LINE_GENERATE** message.
RTGenerateTone If the request succeeds, TAPI will be notified about the tone generation completion through a **LINE_GENERATE** message.

Constructor and Destructor

CTSPICallAppearance	Constructor for the call object.
~CTSPICallAppearance	Destructor for the call object.

Operations - Public Members

AddDeviceClass	Adds a new device class structure to the **LINECALLINFO** structure.
AttachCall	This attaches a call appearance to this call appearance for conferencing or consultation.
CopyCall	Copy an existing call's information into the call object.
CreateCallHandle	Notify TAPI about a call object and obtain a TAPI call handle for the object.
CreateConsultationCall	Creates a new call object and attaches it to the current object as a *consultation* call.
DecRefCount	Decrement the reference count on this call object.
DetachCall	This method removes the association between two call appearances.
GetAddressOwner	Return the owner **CTSPIAddressInfo** object.
GetAttachedCall	Return the pointer to the **CTSPICallAppearance** which is attached to this call. This is used for consultation call management within the library. Anytime a consultation call is created, it is attached to the call it was created for. It is automatically detached when either call goes **Idle**.
GetCallerIDInformation	Returns the **CALLIDENTIFIER** for caller-id.
GetCalledIDInformation	Returns the **CALLIDENTIFIER** for called-id.
GetCallHandle	Return the opaque TAPI handle for this call appearance.
GetCallHub	Returns the call hub object associated with this calls call-id.
GetCallID	Returns the call-id associated with this call object.
GetCallInfo	Return a pointer to the **LINECALLINFO** structure. The pointer may be used to modify any of the data within the structure *except* offset fields.
GetCallState	Return the current callstate (**LINECALLSTATE_**)

	from the **LINECALLSTATUS** record.
GetCallStatus	Return a pointer to the **LINECALLSTATUS** structure. The pointer may be used to modify any of the data within the structure *except* offset fields.
GetCallType	Return the call type (**CALLTYPE_**). This is setup by the library to identify consultant and conference calls on an address.
GetChargingInformation	Returns the associated ISDN charging information with this call.
GetConnectedIDInformation	Returns the **CALLIDENTIFIER** for connected-id.
GetConferenceOwner	Returns the conference owner for this call.
GetConsultationCall	Returns the consultation call associated with this call – or the original call if the current object is a consultation call.
GetDeviceClass	Returns a **DEVICECLASS** structure from the **LINECALLINFO** structure.
GetHiLevelCompatibility Information	Returns the associated ISDN high-level compatibility information for the active call object.
GetLineOwner	Return the **CTSPILineConnection** this call is part of.
GetLowLevelCompatibility Information	Returns the ISDN low-level compatibility information for the active call object.
GetPartiallyDialedDigits	Return the series of digits which have been processed by the service provider for this call but not yet sent to the switch.
GetRedirectingIDInformation	Returns the **CALLIDENTIFIER** for redirecting-id.
GetRedirectionIDInformation	Returns the **CALLIDENTIFIER** for redirection-id.
GetReceivingFlowSpec	Returns the Quality Of Service WinSock FLOWSPEC structure associated with this call.
GetSendingFlowSpec	Returns the Quality Of Service WinSock FLOWSPEC structure associated with this call.
GetShadowCall	Returns the other side of this phone conversation.
HasBeenDeleted	Return whether the call has been deleted by TAPI and is not longer valid.
IncRefCount	Increment the reference count on the object.
IsOutgoingCall	Returns whether this call was created as part of a **lineMakeCall** request.
IsRealCall	Return whether the call is associated with a physical call on the switch.
MarkReal	Mark a call as *real* – i.e. being associated with a call on the switch.
OnDigit	This method should be called when a digit (DTMF or PULSE) is detected on the call.
OnDetectedNewMediaModes	This method should be called when a new media mode is detected on the call.
OnReceivedUserUserInfo	This method should be called when User to User information is received from the network. The data is

	copied into local storage and will be automatically returned in the **LINECALLINFO** record. TAPI is notified.
OnTone	This method should be called when a tone (composed of a set of frequencies) is detected on the call.
ReceiveMSPData	TAPI 3.0 event which receives information from the related media service provider driver.
ReleaseUserUserInfo	Releases the user-user information associated with the call. If there is additional information in the queue then the next block is posted to the **LINECALLINFO** structure.
RemoveDeviceClass	This removes a device class structure from the **LINECALLINFO** structure. It must have been placed there used **AddDeviceClass**.
SetBearerMode	This method should be called to change the bearer mode of the call. TAPI will be notified.
SetCalledIDInformation	This method fills in the called information. TAPI is notified.
SetCallerIDInformation	This method fills in the caller information. TAPI is notified.
SetCallFeatures	This method adjusts the current call features available to the call.
SetCallFeatures2	This method adjusts the current call features available to the call.
SetCallID	This method can be called to change the CALLID field of the call appearance. This may be used to store service provider information about the call. It is not used by the library. TAPI is notified.
SetCallOrigin	This method changes the call origin for the call appearance. TAPI is notified.
SetCallParametersFlag	This method should be called to change the current call parameters. The current call parameters are overwritten. This should be called by the service provider to support **lineSetSecure**. TAPI is notified
SetCallReason	This method changes the call reason for this call appearance. TAPI is notified.
SetCallState	This method should be called to change the call state of the call. All call state changes should come through this method. TAPI is notified.
SetCallData	This sets the 32-bit DWORD value associated with the call. TAPI is notified.
SetCallTreatment	This sets the current call treatment. TAPI is notified.
SetCallType	Sets the internal call type. This is used to identify consultant and conference calls.
SetChargingInformation	Associates ISDN charging information with the call.
SetConferenceOwner	Set the conference owner for this call.

SetConnectedIDInformation	Sets the connected id information for this call.
SetConsultationCall	Associates a consultation call object which has already been created to the current object.
SetDataRate	This method should be called to change the data rate of the call. TAPI will be notified.
SetDestinationCountry	This method changes the destination country for this call appearance. TAPI is notified.
SetDialParameters	This method sets the current dialing parameters for the call appearance. TAPI Is notified.
SetDigitMonitor	This method establishes the digit monitor process for this call.
SetHiLevelCompatibility Information	Associates ISDN high-level compatibility information with the active call object.
SetLowLevelCompatibility Information	Associates ISDN low-level compatibility information with the active call object.
SetMediaControl	This is called when a **lineSetMediaControl** request completes successfully.
SetMediaMonitor	Changes the current set of media modes being monitored.
SetRedirectionIDInformation	This method fills in the redirection id information. TAPI is notified.
SetRedirectingIDInformation	This method fills in the redirecting id information. TAPI is notified.
SetRelatedCallID	This method is used to change the related CALLID field of the call appearance. This is used during conference calls and consultant calls to relate the call to another call object. **DO NOT CHANGE THIS FIELD**.
SetTerminalModes	This method is called to change the terminal modes of the call appearance. This should be invoked once the device has completed a **lineSetTerminal** request. This will automatically be called for any changes to the address or line. TAPI is notified.
SetTrunkID	This method sets the TRUNK ID of the call appearance. TAPI is notified.

Operations - Static Members

IsActiveCallState	Returns whether the supplied callstate is in an active state according to TAPI rules.
IsConnectedCallState	This method returns whether the supplied callstate is *connected* and is taking up bandwidth on the address.

Operations - Protected Members

AddAsynchRequest	Allocate and add a new request to our device asynchronous request list.
CompleteDigitGather	This method is called to end a digit gathering process.
DeleteToneMonitorList	This method is used to delete the active tone monitor list from the call.
SetCallHandle	This method is used to assign the TAPI call handle to the call object in the two-phase construction for this object.

Overridables – Public Members

RecalcCallFeatures	This is called by TSP++ to recalculate the call features for the call object.

Overridables - Protected Members

DestroyObject	This method is called to destroy the call object. The default implementation moves the call object to a global array for future destruction (delayed deletion).
Init	This is called directly after the constructor to initialize the various fields of the **LINECALLINFO** and **LINECALLSTATUS** structures.
OnCallInfoChange	This method is called whenever any data within the **LINECALLINFO** record is changed.
OnCallStatusChange	This method is invoked each time a field in our **LINECALLSTATUS** record changes.
OnMediaControl	This method is called when a media control event is detected. This is only used when media control monitoring is initiated via **lineSetMediaControl**. This generates an asynchronous request.
OnRelatedCallStateChange	This method is called when any call appearance which is related to this call appearance (through the **dwRelatedCallID** field in the **LINECALLINFO** record) changes call states.
OnRequestComplete	This method is called when a request is completed and is associated with this call.
OnShadowCallStateChange	Notification that the other side of this conversation changed its call-state.
OnTerminalCountChanged	The address object invokes this when the count of terminals has changed at the line level. This adds or removes a terminal from the terminal id list.
OnInternalTimer	This is periodically called to check event lists.
OnToneMonitorDetect	This method is called when a tone event that matches a

monitor record is detected. This is only used when a tone list is set up via **lineMonitorTones**. This generates a TAPI notification.

Overridables - TAPI Methods

Accept	This is called by the **lineAccept** method. Default implementation issues a **RTAccept** request.
Answer	This is called by the **lineAnswer** method. Default implementation issues a **RTAnswer** request.
BlindTransfer	This is called by the **lineBlindTransfer** method. Default implementation issues a **RTBlindTransfer** request.
Close	This is called by the **lineClose** method. Default implementation deletes the call appearance.
CompleteCall	This is called by the **lineCompleteCall** method. Default implementation issues a **REQUEST_COMPLETECALL** request.
DevSpecific	This is called by the **lineDevSpecific** function. Default implementation is to return **Not Supported**.
Dial	This is called by the **lineDial** method. Default implementation issues a **RTDial** request.
Drop	This is called by the **lineDrop** method. Default implementation issues a **RTDropCall** request.
GatherCallInformation	This is called by the **lineGetCallStatus** method. This is handled completely within the library.
GatherDigits	This is called by the **lineGatherDigits** method. This is handled completely within the library as long as the **OnDigit** method is called when digits are detected.
GatherStatusInformation	This is called by the **lineGetCallInfo** method. This is handled completely within the library.
GenerateDigits	This is called by the **lineGenerateDigits** method. Default implementation issues a **RTGenerateDigits** request.
GenerateTone	This is called by the **lineGenerateTone** method. Default implementation issues a **RTGenerateTone** request.
GetCallIDs	TAPI 3.0 function to retrieve call-hub related information for a given call object.
GetID	This method is called for **lineGetID**. Default implementation returns **Not Supported**..
Hold	This is called by the **lineHold** method. Default implementation issues a **RTHold** request.
MakeCall	This is called by the **lineMakeCall** method. Default implementation issues a **RTMakeCall** request.
MonitorDigits	This is called by the **lineMonitorDigits** method. This is

	handled within the library if the **OnDigit** method is called when a digit is detected.
MonitorMedia	This is called by the **lineMonitorMedia** method. This is handled within the library if the **OnDetectedNewMediaModes** method is called when new media modes are seen.
MonitorTones	This is called by the **lineMonitorTones** method. This is handled within the library if the **OnTone** method is called when a tone is detected.
Park	This is called by the **linePark** method. Default implementation issues a **RTPark** request.
Pickup	This is called by the **linePickup** method. Default implementation issues a **RTPickup** request.
Redirect	This is called by the **lineRedirect** method. Default implementation issues a **RTRedirect** request.
ReleaseUserUserInfo	This is called by the **lineReleaseUserUserInfo** method. Default implementation deletes any existing user to user information reported by **OnReceivedUserUserInformation**.
Secure	This is called by the **lineSecure** method. Default implementation issues a **RTSecureCall** request.
SendUserUserInfo	This is called by the **lineSendUserUserInfo** method. Default implementation issues a **RTSendUserInfo** request.
SetAppSpecificData	This method is called to change the application specific field of the call. This is called in response to a **lineSetAppSpecific** call. TAPI is notified.
SetCallData	This is called by the **lineSetCallData** method. Default implementation issues a **RTSetCallData** request. The values are placed into the call appearance automatically when the request completes with a zero return code.
SetCallTreatment	This is called by the **lineSetCallTreatment** method. Default implementation issues a **RTSetCallTreatment** request. The values are placed into the call appearance automatically when the request completes with a zero return code.
SetCallParams	This is called by the **lineSetCallParams** method. Default implementation issues a **RTSetCallParams** request. The values are placed into the call appearance automatically when the request completes with a zero return code.
SetMediaMode	This is called by the **lineSetMediaMode** method. This changes the **dwMediaMode** value in the **CALLINFO** record.
SetQualityOfService	This is called by the **lineSetQualityOfService** method. Default implementation issues a

	RTSetQualityOfService request. The values are placed into the call appearance automatically when the request completes with a zero return code.
SwapHold	This is called by the **lineSwapHold** method. Default implementation issues a **RTSwapHold** request.
Unhold	This is called by the **lineUnhold** method. Default implementation issues a **RTUnhold** request.
Unpark	This is called by the **lineUnpark** method. Default implementation issues a **RTUnpark** request.

CTSPICallAppearance::Accept

virtual LONG Accept(DRV_REQUESTID *dwRequestID***,**
LPCSTR *lpszUserUserInfo***, DWORD** *dwSize***);**

dwRequestID	Asynchronous request ID for this request.
lpszUserUserInfo	User to User information to send with the **Accept**.
dwSize	Size of the user to user information block.

Remarks

This method is called in response to an application calling **lineAccept**. The **TSPI_lineAccept** function must be exported from the provider. The method is used to accept an incoming offering call from the switch. On some PBX switch systems, the station receives the offering call before any alerting is performed on the station. The accept method allows the station to begin to ring or notify the user in some other fashion. The class library builds a **RTAccept** packet and inserts it into the request queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::AddAsynchRequest

Protected

int AddAsynchRequest (CTSPIRequest* *pRequest***)**

pRequest	Request to insert into the line list.

Remarks

This method inserts a new request into the pending line request list. If there are no running requests in the list, then this request is actually started and given the opportunity to run the **STATE_INITIAL** processing before this method returns.

By default, this method inserts the request at the end of the line request array.

Return Value

Position that the given request was inserted in the line request list.

CTSPICallAppearance::AddDeviceClass

int AddDeviceClass (LPCTSTR *pszClass***, DWORD** *dwData***);**

int AddDeviceClass (LPCTSTR *pszClass***, HANDLE** *hHandle***,**
LPCTSTR *lpszBuff***);**

int AddDeviceClass (LPCTSTR *pszClass***, HANDLE** *hHandle***, LPVOID** *lpBuff***,**
DWORD *dwSize***);**

int AddDeviceClass (LPCTSTR *pszClass***, DWORD** *dwFormat***, LPVOID** *lpBuff***,**
DWORD *dwSize***, HANDLE** *hHandle* **= INVALID_HANDLE_VALUE);**

int AddDeviceClass (LPCTSTR *pszClass***, LPCTSTR** *pszBuff***,**
DWORD *dwType* **= -1L);**

pszClass	Device class to add/update for (e.g. "tapi/line", etc.)
dwData	DWORD data value to associate with device class.
hHandle	Win32 handle to associate with device class.
lpszBuff	Null-terminated string to associate with device class.
lpBuff	Binary data block to associate with device class
dwSize	Size of the binary data block
dwType	**STRINFORMAT** of the **lpszBuff** parameter.

Remarks

This method adds an entry to the device class list, associating it with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEDEVCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

Index array of added structure.

CTSPICallAppearance::Answer

virtual LONG Answer(DRV_REQUESTID *dwReq***,**
LPCSTR *lpszUserUserInfo***, DWORD** *dwSize***);**

dwRequestID	Asynchronous request ID for this request.
lpszUserUserInfo	User to User information to send with the **Answer**.
dwSize	Size of the user to user information block.

Remarks

This method is called in response to an application calling **lineAnswer**. The **TSPI_lineAnswer** function must be exported from the provider. The method is used to pickup an incoming offering call from the switch. The class library verifies that the call is in the **LINECALLSTATE_OFFERING** call-state, builds a **RTAnswer** packet and inserts it into the request queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::AttachCall

void AttachCall(CTSPICallAppearance* *pCall***);**

pCall	Call to attach this call.

Remarks

This method associates two call appearances together using an internal call object field. This *attachment* is used to keep track of consultation calls relationships created for various events such as transfers, forwards, or conferences. The attached call may be retrieved using **CTSPICallAppearance::GetAttachedCall**.

Anytime the call state for an attached call changes, the other call is notified using the **CTSPICallAppearance::OnRelatedCallStateChange** method.

If either call is deleted, the attachment is automatically broken.

CTSPICallAppearance::BlindTransfer

virtual LONG BlindTransfer(DRV_REQUESTID *dwRequestId***,**
TDialStringArray* *parrDestAddr***, DWORD** *dwCountryCode***);**

dwRequestID	Asynchronous request ID for this request.
parrDestAddr	Address to transfer the call to.
dwCountryCode	Country code for the above dialable address.

Remarks

This method is called in response to an application calling **lineBlindTransfer**. The **TSPI_lineBlindTransfer** function must be exported from the provider. The method is used to perform an unsupervised transfer for an existing call on the switch. The class library verifies that the call is in the **LINECALLSTATE_CONNECTED** state and builds a **RTBlindTransfer** packet and inserts it into the request queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::Close

virtual LONG Close();

Remarks

This method is called in response to all applications that have references to a call destroying the handles through **lineDeallocateCall**. It is invoked through the **TSPI_lineCloseCall** function, which must be exported from the provider. The method deallocates the call object and removes it from the address owner array. After this call is complete, the current object will be invalid and should not be referenced.

Return Value

TAPI error code or zero if the call was deallocated.

CTSPICallAppearance::CompleteCall

virtual LONG CompleteCall (DRV_REQUESTID *dwRequestId***,**
LPDWORD *lpdwCompletionID***, RTCompleteCall*** *lpCompCall***);**

dwRequestID	Asynchronous request ID for this request.
lpdwCompletionID	Completion ID to assign to this request.
lpCompCall	Data structure associated with this request.

Remarks

This method is called in response to an application calling **lineCompleteCall**. It requires that the service provider export the **TSPI_lineCompleteCall** function. It is used to specify how a call that could not be connected normally should be completed instead. The network or switch may not be able to complete a call because the network resources are busy, or the remote station is busy or doesn't answer.

If the service provider completes the method successfully, the line object will copy the information associated with the completion request into an internal array.

Once the switch acknowledges the completion request and sends back an offering call representing the newly completed call, the service provider should use the **CTSPILineConnection::FindCallCompletionRequest** method and pass the appropriate completion identifier within the **RTCompleteCall** data structure into the **CTSPIAddressInfo:CreateCallAppearance** method so that TAPI knows that it is a completion request. In addition, the call reason should be either the default, or specifically set to **LINECALLREASON_CALLCOMPLETION**.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::CompleteDigitGather

void CompleteDigitGather (DWORD *dwReason***);**

dwReason	Reason that the digit gathering process is being stopped (**LINEGATHERTERM_xx**).

Remarks

This method completes a digit gathering session and deletes and resets the digit gather. It is called when a termination digit is found, the buffer is full, or a gather request is cancelled.

CTSPICallAppearance::CopyCall

void CopyCall(CTSPICallAppearance* *pCall***, bool** *fShadowCall***=true);**

pCall	Call object to copy information from
fShadowCall	Whether this call is a *shadow* of the passed call.

Remarks

This method copies an existing call into the current call object. Most of the **LINECALLINFO** fields are copied.

If the *fShadowCall* field is non-zero, then the passed call is considered a shadow call of the current call object and both calls will be a part of the call hub for the call-id. If the value is zero then the passed call object's call-id field is set to zero and the current call *replaces* the passed call in the call hub object.

CTSPICallAppearance::CreateCallHandle

bool CreateCallHandle();

Remarks

This method is used to notify TAPI about an existing call in the service provider that was created when the line owner was not open (and therefore not capable of receiving telephony events).

It causes TAPI to recognize the call and create a **HTAPICALL** handle for it. The call-state is automatically updated in the TAPI record. Once this method finished, applications can see the call on the line owner.

This method is used by TSP++ to notify TAPI of existing calls when a line owner opens a line and the media mode is established. It generally does not need to be invoked by the derived provider code.

Return Code

The function returns whether the call was successfully registered with TAPISRV.

CTSPICallAppearance::CreateConsultationCall

CTSPICallAppearance* CreateConsultationCall (HTAPICALL *htCall***,**
DWORD *dwCallParamFlags***=0);**

htCall	Existing TAPI call handle. If this is NULL, TAPI will be asked for a new call handle.
dwCallParamFlags	Call parameters to associate with this call.

Remarks

This method creates a new call object and associates the existing call with the call as a *consultation call*. This is normally used in multi-step conferencing or transfer operations where the switch places an existing call on hold and puts the handset in the dialtone state.

The consultation relationship is two-way; either call can locate the other through the **GetConsultationCall** method. In addition, if the consultationc all goes *Idle*, the original call is notified through the virtual **OnConsultationCallIdle** method.

Return Value

New call appearance object with the call type of *Consultant*.

CTSPICallAppearance::CTSPICallAppearance

CTSPICallAppearance();

Remarks

This is the constructor for the call object. Any derived class cannot have any parameters as it is created within the class library.

CTSPICallAppearance::~CTSPICallAppearance

~CTSPICallAppearance();

Remarks

This is the destructor for the call object.

CTSPICallAppearance::DecRefCount

void DecRefCount();

Remarks

This method decrements the reference count for the call object. If the reference count hits zero, the call is moved to the deleted array (and deleted by the TSP++ internal interval timer).

It should generally not be called by the derived provider as TSP++ manages the call lifetime automatically.

CTSPICallAppearance::DeleteToneMonitorList

Protected

void DeleteToneMonitorList();

Remarks

This internal method is used to delete the list of tones which are being monitored for by TAPI.

CTSPICallAppearance::DestroyObject

Protected

virtual void DestroyObject();

Remarks

This internal method is used to move the now-idle call object into a global future deletion list. This is done to keep the memory around past the lifetime of the TAPI call handle in case of heavy load where some events arrive after the call object has been destroyed by TAPI.

CTSPICallAppearance::DetachCall

void DetachCall();

Remarks

This internal method is used to remove the attachment from this call appearance to another. The calls were attached using **CTSPICallAppearance::AttachCall**. This call attachment is used in consultation and conference calls.

CTSPICallAppearance::DevSpecific

virtual LONG DevSpecific(DRV_REQUESTID *dwRequestID*,
LPVOID *lpParam*, **DWORD** *dwSize*);

dwRequestID	Asynchronous request ID associated with this request.
lpParams	Optional pointer to a parameter block
dwSize	Size of the above parameter block

Remarks

This method is called by the library when an application calls **lineDevSpecific** and identifies this call object in the **HCALL** parameter. The **TSPI_lineDevSpecific** function must be exported from the provider. This method enables service providers to provide access to features not typically offered by TAPI. The meaning of these extensions are device specific and known only to the service provider and applications which are written to take advantage of them. This method is not handled directly by the library and is provided simply for overriding purposes.

Return Value

You must override this method to provide device-specific functionality. The library returns **LINEERR_OPERATIONUNAVAIL** if you do not override this method.

CTSPICallAppearance::Dial

virtual LONG Dial (DRV_REQUESTID *dwRequestID***,**
TDialStringArray* *parrAddresses***, DWORD** *dwCountryCode***);**

dwRequestID	Asynchronous request ID for this request.
parrAddresses	Dialable addresses to dial.
dwCountryCode	Country code to use for addresses.

Remarks

This method is called in response to an application calling **lineDial**. It requires that the service provider export the **TSPI_lineDial** function. It is used on existing call appearances to continue the dialing process after the **TSPI_lineMakeCall** is completed.

The class library validates that the call is not **Idle** or **Disconnected**, adjusts the caller id and called id information to include the digits if the call is in a known dialing state, and submits a **RTDial** request on the line.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::Drop

virtual LONG Drop(DRV_REQUESTID *dwRequestId***=0,**
LPCSTR *lpszUserUserInfo* **= NULL, DWORD** *dwSize* **= 0);**

dwRequestID	Asynchronous request ID for this request.
lpszUserUserInfo	User to user information passed with the drop request.
dwSize	Size of the user to user information block.

Remarks

This method is called in response to an application calling **lineDrop**. TAPI invokes the **TSPI_lineDial** function which must be exported. It is used to drop a connection on a call appearance.

The class library submits a **REQUEST_DROP** request if the call appearance isn't already in the **Idle** state.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::GatherCallInformation

virtual LONG GatherCallInformation (LPLINECALLINFO *lpCallInfo***);**

lpCallInfo TAPI **LINECALLINFO** structure to fill.

Remarks

This method is called in response to an application calling **lineGetCallInfo**. TAPI invokes the **TSPI_lineGetCallInfo** function which must be exported. It is used to return information about the call appearance to TAPI and invoking applications.

The class library gathers all the call information from internal data structures and the internal **LINECALLINFO** structure maintained within the call object. This method may be overridden by the service provider if device specific features are implemented by the provider.

The derived provider can adjust the return values by using the provider **SetXXX** methods or using the **CTSPICallAppearance::GetCallInfo** method and adjusting the returning structure.

Return Value

TAPI error code or zero if the method was successful.

CTSPICallAppearance::GatherDigits

virtual LONG GatherDigits (TSPIDIGITGATHER* *lpGather***);**

lpGather Data structure with information for request.

Remarks

This method is called in response to an application calling **lineGatherDigits**. The service provider must export the **TSPI_lineGatherDigits** function. It is used to allocate and assign a buffer for all DTMF or pulse digits noticed by the service provider on the media stream.

It Initiates the buffered gathering of digits on the specified call. The application specifies a buffer in which to place the digits and the maximum number of digits to be collected.

The class library implements this method completely – the service provider simply needs to export the **TSPI_lineGatherDigits** function and then call the **CTSPICallAppearance::OnDigit** method when any digit is seen on the media stream.

Return Value

TAPI error code or zero if the method was successful.

CTSPICallAppearance::GatherStatusInformation

virtual LONG GatherStatusInformation(LPLINECALLSTATUS *lpCallStatus***);**

lpCallStatus	TAPI **LINECALLSTATUS** structure to fill.

Remarks

This method is called in response to an application calling **lineGetCallStatus**. TAPI will call the **TSPI_lineGetCallStatus** function which must be exported by the service provider. It is used by the application to see the current status features and abilities of the call.

The class library implements this method completely and returns all the known information about the call from the internal data structures. The derived provider can adjust the return values by using the provider **SetXXX** methods or using the **CTSPICallAppearance::GetCallStatus** method and adjusting the returning structure.

Return Value

TAPI error code or zero if the method was successful.

CTSPICallAppearance::GenerateDigits

virtual LONG GenerateDigits (DWORD *dwEndToEndID***,**
DWORD *dwDigitMode***, LPCTSTR** *lpszDigits***, DWORD** *dwDuration***);**

dwEndToEndID	Unique identifier for this digit generation.
dwDigitMode	**LINEDIGITMODE_xxx** value.
lpszDigits	NULL-terminated List of digits to generate.
dwDuration	Requested millisecond pause between digits.

Remarks

This method is called in response to an application calling **lineGenerateDigits**. The service provider must export the **TSPI_lineGenerateDigits** function.

The method initiates the generation of the specified digits on the specified call as inband tones using the specified signaling mode. Invoking this method with a NULL value for *lpszDigits* aborts any digit generation currently in progress. An application that invokes lineGenerateDigits while digit generation is in progress aborts the current digit generation and initiates the generation of the most recently specified digits.

The class library validates the passed parameters, removes any pending **RTGenerateDigits** request, and creates and inserts a new **RTGenerateDigits** request into the line queue.

Return Value

TAPI error code or zero if the method added the **RTGenerateDigits** request.

CTSPICallAppearance::GenerateTone

virtual LONG GenerateTone (DWORD *dwEndToEndID***,**
DWORD *dwToneMode***, DWORD** *dwDuration***,**
DWORD *dwNumTones***, LPLINEGENERATETONE** *lpTones***);**

dwEndToEndID	Unique identifier for this generate tone request.
dwToneMode	**LINETONEMODE_xxx** constant for the type.
dwDuration	Duration of the tone in milliseconds.
dwNumTones	Number of tones passed.
lpTones	Array of passed tones.

Remarks

This method is called in response to an application calling **lineGenerateTone**. The service provider must export the **TSPI_lineGenerateTone** function.

This method generates the specified inband tone over the specified call. Invoking this method with a zero for *dwToneMode* aborts the tone generation currently in progress on the specified call. An application that invokes lineGenerateTone while tone generation is in progress aborts the current tone generation and initiates the generation of the newly specified tone.

The class library validates the passed parameters, removes any pending **RTGenerateTone** request, and creates and inserts a new **RTGenerateTone** request into the line queue.

Return Value

TAPI error code or zero if the method added the **RTGenerateTone** request.

CTSPICallAppearance::GetAddressOwner

CTSPIAddressInfo* GetAddressOwner() const;

Remarks

This method may be used by the service provider to determine which address object the call belongs to.

Return Value

The **CTSPIAddressInfo** object which owns the call.

CTSPICallAppearance::GetAttachedCall

CTSPICallAppearance* GetAttachedCall();

Remarks

This method may be used by the service provider to determine if any call appearances are attached to this call. Attached calls are used internally in the library to maintain connections between consultation and conference calls.

Return Value

The **CTSPICallAppearance** object which is attached to the current call or NULL if there is no attached call.

CTSPICallAppearance::GetCallHandle

HTAPICALL GetCallHandle() const;

Remarks

This method may be used by the service provider to get the TAPI opaque handle which represents this call in the TAPI system. For more information on opaque handles, see the section on *TAPI handles* in the beginning of the manual.

Return Value

The opaque TAPI handle which represents the current call object in the TAPI system DLLs.

CTSPICallAppearance::GetCallHub

CTSPICallHub* GetCallHub() const;

Remarks

This method returns the call hub object that maintains the relationship between different call objects that share the call-id of this call. For more information on this object, see the reference section on *CTSPICallHub*.

Return Value

Pointer to the call hub – this will always be valid as long as the call-id for the call is non-zero.

CTSPICallAppearance::GetCalledIDInformation

const CALLIDENTIFIER& GetCalledIDInformation() const;

Remarks

This method returns the current called-id settings for the call. See the reference section on *Data Structures* for the definition of **CALLIDENTIFIER**.

Return Value

A constant reference to the called-id information for the call.

CTSPICallAppearance::GetCallerIDInformation

const CALLIDENTIFIER& GetCallerIDInformation() const;

Remarks

This method returns the current caller-id settings for the call. See the reference section on *Data Structures* for the definition of **CALLIDENTIFIER**.

Return Value

A constant reference to the caller-id information for the call.

CTSPICallAppearance::GetCallID

DWORD GetCallID() const;

Remarks

This method returns the call-id associated with the call. This is the same as calling **GetCallInfo()->dwCallID**.

Return Value

The 32-bit call-id assigned to this call objet

CTSPICallAppearance::GetCallIDs

virtual LONG GetCallIDs(LPDWORD *lpdwAddressID***,LPDWORD** *lpdwCallID***,**
LPDWORD *lpdwRelatedCallID***)**

lpdwAddressID	Returning id assigned to the address owner.
lpdwCallID	Returning assigned call-id
lpdwRelatedCallID	Returning assigned related call id.

Remarks

This method is used by TAPI 3.0 to quickly gather information about call-hubs related for this call object. This information can also be gathered from the **LINECALLINFO** structure, but this method executes much faster. It is only used if the TSP negotiates to TAPI 3.0 or above and exports the **TSPI_lineGetCallIDs** function.

Return Value

TAPI error code or zero if successful.

CTSPICallAppearance::GetCallInfo

LPLINECALLINFO GetCallInfo();
const LPLINECALLINFO GetCallInfo() const;

Remarks

This method returns the structure that maps the current call information for the call object. Any information that is not an offset/size buffer may be modified or read from this structure.

If the structure is modified using this method, TAPI is not automatically notified.

Return Value

Pointer to the structure maintained by the call object.

CTSPICallAppearance::GetCallState

DWORD GetCallState() const;

Remarks

This method returns the current TAPI call state of the call object. The call state reflects the current state of the connection between this call and the destination or source. The value returned from here will always match the current value in the **LINECALLSTATE.dwCallState** element. The call state starts at

LINECALLSTATE_UNKNOWN and may be adjusted by the service provider using the **CTSPICallAppearance::SetCallState**.

Return Value

The current TAPI call state.

CTSPICallAppearance::GetCallStatus

LPLINECALLSTATUS GetCallStatus();
const LPLINECALLSTATUS GetCallStatus() const

Remarks

This method returns the structure which maps the current status for the call object. Any information which is not an offset/size buffer may be modified or read from this structure.

If the structure is modified using this method, TAPI is not automatically notified.

Return Value

Pointer to the structure maintained by the call object.

CTSPICallAppearance::GetCallType

int GetCallType() const;

Remarks

This method returns the type of call this call object represents.

Call Type	Description
Normal	Normal call on the switch.
Conference	Conference call, can be cast to a **CTSPIConferenceCall**.
Consultant	Consultant call – attached to a normal call, represents a call created on the switch for a pending transfer/forward/conference event.

Return Value

Type of call (enumerated type from **CTSPICallAppearance**) for this object.

CTSPICallAppearance::GetChargingInformation

const SIZEDDATA GetChargingInformation() const;

Remarks

This method returns the ISDN charging information associated with the call object.

Return Value

Pointer to a **SIZEDDATA** structure containing the information.

CTSPICallAppearance::GetConferenceOwner

CTSPIConferenceCall* GetConferenceOwner() const;

Remarks

This method returns the conference call that this call is part of.

Return Value

Pointer to the conference owner, or NULL if the call is not currently part of a conference.

CTSPICallAppearance::GetConnectedIDInformation

const CALLIDENTIFIER& GetConnectedIDInformation() const;

Remarks

This method returns the current connected-id settings for the call. See the reference section on *Data Structures* for the definition of **CALLIDENTIFIER**.

Return Value

A constant reference to the connected-id information for the call.

CTSPICallAppearance::GetConsultationCall

CTSPICallAppearance* GetConsultationCall() const;

Remarks

This method locates any consultation call associated with the call object, or locates the original call from a consultation call. The consultation call will return *Consultant* for the **GetCallType** method.

Return Value

The call object associated with the given call.

CTSPICallAppearance::GetDeviceClass

DEVICECLASSINFO* GetDeviceClass(LPCTSTR *pszClass***);**

pszClass Device class to search for (e.g. “tapi/line”, etc.)

Remarks

This method returns the device class structure associated with the specified class name. If you associate class information with an object, it is automatically reported back through the **LINEDEVCAPS** structure and returned using **TSPI_lineGetID**.

Return Value

DEVICECLASSINFO structure which represents the specified text name. NULL if no association has been performed.

CTSPICallAppearance::GetHiLevelCompatibilityInformation

const SIZEDDATA GetHiLevelCompatibilityInformation() const;

Remarks

This method returns the ISDN high-level compatibility information associated with the call object.

Return Value

Pointer to a **SIZEDDATA** structure containing the information.

CTSPICallAppearance::GetID

virtual LONG GetID (TString& *strDevClass***, LPVARSTRING** *lpDeviceID***, HANDLE** *hTargetProcess***);**

strDeviceClass	Device class which is being asked for.
lpDeviceID	Returning device information.
hTargetProcess	Process requesting data.

Remarks

This method is called in response to an application calling **lineGetID** and specifying a specific call handle. It is used to return device information to the application based on a string device class key. TSP++ implements this method internally and returns any device information which was added using **CTSPICallAppearance::AddDeviceClass**.

Any handle which was given to the **AddDeviceClass** method is automatically duplicated in the given process for TAPI 2.x using the **DuplicateHandle** Win32 function.

Return Value

TAPI error code or zero if the method was successful.

CTSPICallAppearance::GetLineOwner

CTSPILineConnection* GetLineOwner() const;

Remarks

This method returns the owning line object for this call object.

Return Value

Pointer to the line object that owns this call object.

CTSPICallAppearance::GetLowLevelCompatibilityInformation

const SIZEDDATA GetLowLevelCompatibilityInformation() const;

Remarks

This method returns the ISDN low-level compatibility information associated with the call object.

Return Value

Pointer to a **SIZEDDATA** structure containing the information.

CTSPICallAppearance::GetPartiallyDialedDigits

TString& GetPartiallyDialedDigits();

Remarks

This method returns the current string representing the partial dialing sequence sent to the service provider for this call.

This can be used by the derived provider to allow the provider to support progressive dialing on telephony hardware that does not support it.

Return Value

Reference to the string object containing the current set of dialed digits.

CTSPICallAppearance::GetReceivingFlowSpec

const SIZEDDATA GetReceivingFlowSpec() const;

Remarks

This method returns the Winsock FLOWSPEC information related to the QOS of the receiving media stream.

Return Value

Pointer to a **SIZEDDATA** structure containing the information.

CTSPICallAppearance::GetRedirectingIDInformation

const CALLIDENTIFIER& GetRedirectingInformation() const;

Remarks

This method returns the current redirecting-id settings for the call. See the reference section on *Data Structures* for the definition of **CALLIDENTIFIER**.

Return Value

A constant reference to the redirecting-id information for the call.

CTSPICallAppearance::GetRedirectionIDInformation

const CALLIDENTIFIER& GetRedirectionInformation() const;

Remarks

This method returns the current redirection-id settings for the call. See the reference section on *Data Structures* for the definition of **CALLIDENTIFIER**.

Return Value

A constant reference to the redirection-id information for the call.

CTSPICallAppearance::GetSendingFlowSpec

const SIZEDDATA GetSendingFlowSpec() const;

Remarks

This method returns the Winsock FLOWSPEC information related to the QOS of the sending media stream.

Return Value

Pointer to a **SIZEDDATA** structure containing the information.

CTSPICallAppearance::GetShadowCall

CTSPICallAppearance* GetShadowCall() const;

Remarks

This method returns the other side of the current call conversation. The shadow call is the other call that is in the call hub and shares the call-id.

If there is only one or more than one call in the call hub, this method returns NULL.

Return Value

A pointer to the shadow call for this object, NULL if the method cannot determine it or this is not an in-switch call.

CTSPICallAppearance::HasBeenDeleted

bool HasBeenDeleted() const;

Remarks

This method returns whether the current call has been deallocated by the TAPI server and is therefore, invalid.

Return Value

TRUE if the call object has been deleted (and is awaiting real memory deletion by the TSP). FALSE if the call is active in the Windows telephony session.

CTSPICallAppearance::Hold

virtual LONG Hold (DRV_REQUESTID *dwRequestID***);**

dwRequestID TAPI Asynchronous request ID for the request.

Remarks

This method is called in response to an application calling **lineHold**. The service provider must export the **TSPI_lineHold** function. The class library checks the call state to make sure it is in **Connected**, **Proceeding**, **Dialing**, or **Dialtone**, and submits a **RTHold** request into the line queue.

Return Value

TAPI error code or the request ID if the asynchronous request was added to the queue.

CTSPICallAppearance::IncRefCount

void IncRefCount();

Remarks

This method increments the reference count for the call object. The call will not be deleted until the reference count hits zero.

It should generally not be called by the derived provider as TSP++ manages the call lifetime automatically.

CTSPICallAppearance::Init

Protected

virtual void Init (CTSPIAddressInfo* *pAddress*, **DWORD** *dwBearerMode*,
DWORD *dwRate*, **DWORD** *dwCallParamFlags*,
DWORD *dwOrigin*, **DWORD** *dwReason*, **DWORD** *dwTrunk*,
DWORD dwCompletionID);

pAddress	Address owner for this call.
dwBearerMode	**LINEBEARERMODE_xxx** from the address.
dwRate	Data rate for this call (current rate on address).
dwCallParamFlags	Call parameter flags passed to address.
dwOrigin	**LINECALLORIGIN_xxx** from the address.
dwReason	**LINECALLREASON_xxx** from the address.
dwTrunk	Trunk identifier for call.
dwCompletionID	Completion identifier from a previous **CompleteCall** request.

Remarks

This method is called right after the constructor to initialize the call appearance. The values passed to this method correspond directly to the values passed to the **CTSPIAddressInfo::CreateCallAppearance** method. This is the first phase portion of the call object construction. The call-handle from TAPI (if there is one) is assigned by the **SetCallHandle** method after this function completes.

CTSPICallAppearance::IsActiveCallState

static bool IsActiveCallState(DWORD *dwState*);

dwState	Call state in question.

Remarks

This method can be used by the service provider to determine if the given call state is considered *active* by the class library and TAPI. A call state is considered active if it is not **Idle**, **OnHold** (for any reason), or **Disconnected**.

Return Value

TRUE or FALSE if the call state is considered active.

CTSPICallAppearance::IsConnectedCallState

static bool IsConnectedCallState(DWORD *dwState***);**

dwState Call state in question.

Remarks

This method can be used by the service provider to determine if the given call state is considered *connected* by the class library. A call state is considered active if it is tying up a channel on the line (i.e. no other call can be present due to this call). This would *not* include things such as **OnHold** or **Offering**. The method is subtlety different than **CTSPICallAppearance::IsActiveCallState**, incoming calls are not treated as connected, but they *are* treated as active.

Return Value

TRUE or FALSE if the call state is considered connected.

CTSPICallAppearance::IsOutgoingCall

bool IsOutgoingCall() const;

Remarks

This method returns whether the current call was created in response to a **lineMakeCall** or other outgoing call event.

Return Value

TRUE if the call object was originally created as an outgoing call (vs. an incoming event).

CTSPICallAppearance::IsRealCall

bool IsRealCall();

Remarks

This method returns whether the call object has been dialed on the hardware and represents a physical call on the PBX/ACD system. It does *not* indicate that the call-id field is valid as that may not have been determined yet.

This can be used by the derived provider to allow the provider to support progressive dialing on telephony hardware that does not support it.

This feature must be managed by the derived provider using the **MarkReal** method to change the flag to *not dialed* or *dialed*. Unless the call is specifically marked as a non-dialed call, this method will always return true.

Return Value

TRUE if the call has been dialed on the hardware, FALSE if it is still dialing.

CTSPICallAppearance::MakeCall

virtual LONG MakeCall (DRV_REQUESTID *dwRequestID***,**
TDialStringArray* *parrDialInfo***, DWORD** *dwCountryCode***,**
LPLINECALLPARAMS *lpCallParams***);**

dwRequestID	Asynchronous request ID for this request.
parrDialInfo	Array of **DIALINFO** structures detailing the number(s) to dial.
dwCountryCode	The counry code to assume for this dialing.
lpCallParams	**LINECALLPARAMS** for the newly created call.

Remarks

This method is called in response to an application calling **lineMakeCall**. The service provider must export the **TSPI_lineMakeCall** function.

The method is called by the **CTSPILineConnection::MakeCall** method as part of the new call processing. It sets up the origin, reason and country code within the call object, copies the **LINECALLPARAMS** buffer if it was given by the application, established caller id and called id information, and submits a **RTMakeCall** request to the line queue.

Return Value

TAPI error code or the request ID if the asynchronous request was added to the queue.

CTSPICallAppearance::MarkReal

void MarkReal(bool *fIsReal***=true);**

fIsReal	TRUE/FALSE whether the call is on the hardware.

Remarks

This method changes the flag as to whether the call has been dialed on the hardware yet.

This can be used by the derived provider to allow the provider to support progressive dialing on telephony hardware that does not support it.

See the method **IsRealCall** for more information on this topic.

CTSPICallAppearance::MonitorDigits

virtual LONG MonitorDigits (DWORD *dwDigitModes***);**

dwDigitModes	Digit modes to monitor for.

Remarks

This method is called in response to an application calling **lineMonitorDigits**. The service provider must export the **TSPI_lineMonitorDigits** function.

It enables and disables the unbuffered detection of digits received on the call. Each time a digit of the specified digit mode(s) is detected, a message is sent to the application indicating which digit has been detected. The method is completely implemented within the class library, all the service provider needs to do is call the **CTSPICallAppearance::OnDigit** method each time a digit is detected on the device.

Return Value

TAPI error code or zero if the new digit modes will be monitored for.

CTSPICallAppearance::MonitorMedia

virtual LONG MonitorDigits (DWORD *dwMediaModes***);**

dwMediaModes Media modes to monitor for.

Remarks

This method is called in response to an application calling **lineMonitorMedia**. The service provider must export the **TSPI_lineMonitorMedia** function.

It enables and disables the detection of media modes on the specified call. When a media mode is detected, a message is sent to the application.. The method is completely implemented within the class library, all the service provider needs to do is call the **CTSPICallAppearance::OnDetectedNewMediaModes** method when a new media mode is detected on the media stream.

Return Value

TAPI error code or zero if the new media modes will be monitored for.

CTSPICallAppearance::MonitorTones

virtual LONG MonitorTones (TSPITONEMONITOR* *lpMon***);**

lpMon Data structure with tones to monitor for.

Remarks

This method is called in response to an application calling **lineMonitorTones**. The service provider must export the **TSPI_lineMonitorTones** function.

It enables and disables the detection of inband tones on the call. Each time a specified tone is detected, a message is sent to the application.. The method is completely implemented within the class library, all the service provider needs to do is call the **CTSPICallAppearance::OnTone** method when a new tone is detected on the media stream.

If the tones generated are not specific or do not come in three frequency sets, the provider may override the **CServiceProvider:: MatchTones** method to manage the comparison between detected tones and monitoring tones.

Return Value

TAPI error code or zero if the new tones will be monitored for.

CTSPICallAppearance::OnCallInfoChange

Protected

virtual void OnCallInfoChange (DWORD *dwCallInfo***);**

dwCallInfo	**LINECALLINFO** field which changed.

Remarks

This method is called whenever information in our **LINECALLINFO** record has changed. It notifies TAPI about the change to the call object if necessary. If the service provider changes data directly in the **LINECALLINFO** record using **CTSPICallAppearance::GetCallInfo**, it should invoke this method to tell TAPI.

CTSPICallAppearance:: OnConsultantCallIdle

Protected

virtual void OnConsultantCallIdle(CTSPICallAppearance* *pConsultCall***);**

pConsultCall	Call which has gone idle

Remarks

This notification is called when the consultant call attached to the call object call goes **Idle**, indicating that either party hung up. The result of this is dependant on the switch hardware. The default implementation does nothing.

CTSPICallAppearance::OnDetectedNewMediaModes

void OnDetectedNewMediaModes (DWORD *dwMediaModes***);**

dwMediaModes	New media mode(s) seen on the media stream.

Remarks

The service provider should call this method when a change on the media stream is detected. This would happen if the device supported more than one media mode, and was capable of doing tone detection on the line. This method drives the tone monitoring support in the class library.

Note that the service provider should include *all* media modes which have been detected since this replaces the existing media modes in the **LINECALLSTATUS** structure.

CTSPICallAppearance::OnDigit

void OnDigit (DWORD *dwType***, char** *cDigit***);**
void OnDigit (DWORD *dwType***, TCHAR** *cDigit***);**

dwType	Type of digit detected (**LINEDIGITMODE_xxx**).
cDigit	Specific digit detected.

Remarks

The service provider should call this method when a DTMF or pulsed digit is detected on the connection. This method drives the digit monitoring and gathering support in the class library.

CTSPICallAppearance::OnInternalTimer

Protected

virtual void OnInternalTimer();

Remarks

This internal method is called by the address when our interval timer is set. The class library uses it to check the call digit gathering process to insure that it has not timed out.

CTSPICallAppearance::OnMediaControl

Protected

virtual void OnMediaControl (DWORD *dwMediaControl***);**

dwMediaControl	Media control event which just fired.

Remarks

This notification method is called when a media control event was activated due to a media monitoring event being detected by the class library. The default implementation inserts a **RTSetMediaControl** request into the line queue so the service provider may perform work in the context of its worker thread.

CTSPICallAppearance::OnReceivedUserUserInfo

void OnReceivedUserUserInfo (LPVOID *lpBuff***, DWORD** *dwSize***);**

lpBuff	Buffer received
dwSize	Size of the buffer received in bytes

Remarks

The service provider may call this method if user-to-user information is received by the device from the underlying switch network. The data is copied into an internal buffer and presented to TAPI when requested.

CTSPICallAppearance::OnRelatedCallStateChange

Protected

virtual void OnRelatedCallStateChange (CTSPICallAppearance* *pCall***,**
DWORD *dwState***, DWORD** *dwOldState***);**

pCall	Call appearance which changed state.
dwState	New call state of the call.
dwOldState	Previous call state of the call.

Remarks

This notification method is called whenever a call which is related to this call changes state. The call relationship is made through the **CTSPICallAppearance::AttachCall** and **LINECALLINFO.dwRelatedCallID** fields and is used by conference and consultation calls to relate them to a call appearance.

CTSPICallAppearance::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq*, **LONG** *lResult*);

pReq	Request which has completed
lResult	Final return code for the request.

Remarks

This method is called when a request is completed on the owner address/line. It gives the call object an opportunity to cleanup any data or information associated with the request.

The default implementation of the class library manages several requests:

RTSetTerminal

If the request completes successfully, the terminal information is updated in the address object using the **CTSPIAddressInfo::SetTerminalModes** method.

RTSetCallParams

If the request completes successfully, the bearer mode and dialing parameters are copied into the call object.

REQUST_DROPCALL

If the request completes successfully, all the pending requests for the call are removed from the line queue.

RTSecureCall

If the request completes successfully the **LINECALLPARAMFLAGS_SECURE** flag is added to the **LINECALLINFO.dwCallParamFlags** field and TAPI is notified of the change.

RTSwapHold

If the request completes successfully, the call types between the **onHold** call and the **Connected** call are swapped (consultation for normal).

RTGenerateDigits

If the request completes successfully, then TAPI is told about the request through a **LINE_GENERATE** notification.

RTGenerateTone

If the request completes successfully, then TAPI is told about the request through a **LINE_GENERATE** notification.

RTSetCallData

If the request completes successfully, then the call data associated with the object is updated using **SetCallData** from the request.

RTSetQualityOfService

If the request completes successfully, then the quality of service call data associated with the object is updated using **SetQualityOfService** from the request.

CTSPICallAppearance::OnShadowCallStateChange

protected

virtual void OnShadowCallStateChange(CTSPICallAppearance* *pCall*, **DWORD** *dwState*, **DWORD** *dwCurrState*);

pCall	Call which changed state.
dwState	New state for the given call
dwCurrState	Current state for the given call.

Remarks

This method is called when any of the calls in the call-hub for the current call change their call state. The call state has not been changed in the **LINECALLSTATUS** record yet.

CTSPICallAppearance::OnTerminalCountChanged

Protected

virtual void OnTerminalCountChanged (bool *fAdded*, **int** *iPos*, **DWORD** *dwMode* **= 0L);**

fAdded	Whether the terminal mode was ADDED or REMOVED.
iPos	Position of the affected terminal.
dwMode	Terminal mode which was added or removed.

Remarks

The class library uses this method to synchronize the terminal modes between the line, address and call objects. When the terminal information is adjusted through TAPI using the **lineSetTerminal** function, this method adjusts each of the calls on the address to match the new terminal modes. The terminal mode is either *added* to the existing terminal, or *removed* depending on the first parameter. TAPI is notified of the change through a **LINECALLINFOSTATE_TERMINAL** event notification.

CTSPICallAppearance::OnTone

void OnTone (DWORD *dwFreq1*, **DWORD** *dwFreq2* **= 0, DWORD** *dwFreq3* **= 0);**

Remarks

The derived service provider should call this method when a frequency is detected on the connection represented by this call. It drives the tone monitoring capabilities in the class library. Also, check out the **CServiceProvider::MatchTones** method.

CTSPICallAppearance::OnToneMonitorDetect

virtual void OnToneMonitorDetect (DWORD *dwToneListID***,**
DWORD *dwAppSpecific***);**

dwToneListID	Tone list identifier which was detected
dwAppSpecific	Application specific data passed from TAPI.

Remarks

This method is called when a monitored tone has been detected. It notifies TAPI using the **LINE_MONITORTONE** event.

CTSPICallAppearance::Park

virtual LONG Park (DRV_REQUESTID *dwRequestID***,**
DWORD *dwParkMode***, TDialStringArray*** *parrAddresses***,**
LPVARSTRING *lpNonDirAddress***);**

dwRequestID	TAPI asynchronous request ID.
dwParkMode	**LINEPARKMODE_xxx** type of park requested
parrAddresses	Array of **DIALINFO** structures with information on the requested parking address for a directed park request.
lpNonDirAddress	The returned parking address if a non-directed park is requested.

Remarks

This method is called in response to an application calling **linePark**. The service provider must export the **TSPI_linePark** function. It is used to park an active call at a destination address on the switch. The class library verifies that the call is in the **Connected** state, validates the requested park mode, and submits a **RTPark** asynchronous request.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::Pickup

virtual LONG Pickup (DRV_REQUESTID *dwRequestID***,**
TDialStringArray* *parrDial***, LPCTSTR** *pszGroupID***);**

dwRequestID	TAPI asynchronous request ID.
parrDial	Array of **DIALINFO** structures with address to pickup call from.
pszGroupID	Used for group pickup events.

Remarks

This method is called in response to an application calling **linePickup**. The service provider must export the **TSPI_linePickup** function. It is used to pick up a call alerting at the specified destination address and return a call handle for the picked up call. If the p*arrDial* array is empty, a group pickup is performed. If required by the device capabilities, *pszGroupID* specifies the group ID to which the alerting station belongs..

The class library sets the caller id information appropriately and submits a **RTPickup** request.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPILineConnection::RecalcCallFeatures

virtual void RecalcCallFeatures(DWORD *dwState***=0);**

dwState	Call-state to use for feature determination.

Remarks

This method is called when the call status is changed to re-determine the call's feature set based on the current call state.

It may be overridden to catch all changes to the call state.

CTSPICallAppearance::ReceiveMSPData

Protected

virtual LONG ReceiveMSPData(CMSPDriver* *pMSP***,**
LPVOID *lpData***, DWORD** *dwSize***);**

Remarks

This method receives notifications from the attached media service proider. This event is only used if the TSP negotiates to TAPI 3.0 or above, provides MSP support, and exports the **TSPI_lineReceiveMSPData** function.

CTSPICallAppearance::Redirect

virtual LONG Redirect (DRV_REQUESTID *dwRequestID***,**
TDialStringArray* *parrAddresses***, DWORD** *dwCountryCode***);**

dwRequestID	TAPI asynchronous request ID.
parrAddresses	Address to redirect call to.
dwCountryCode	Country code associated with dialable address.

Remarks

This method is called in response to an application calling **lineRedirect**. The service provider must export the **TSPI_lineRedirect** function. It is used to redirect an offering call to another station or dialable address.

The class library verifies that the call is **Offering**, sets the redirection id information appropriately and submits a **RTRedirect** request.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::ReleaseUserUserInfo

virtual LONG ReleaseUserUserInfo(DRV_REQUESTID *dwRequestID***)**

dwRequestID TAPI asynchronous request ID.

Remarks

This method is called in response to TAPI releasing user-user information that is present in the **LINECALLINFO** record. The service provider must export the **TSPI_lineReleaseUserUserInfo** function. The method generates the **RTReleaseUserInfo** asynchronous request and inserts it into the request list.

Return Value

TAPI result code

CTSPICallAppearance::ReleaseUserUserInfo

void ReleaseUserUserInfo();

Remarks

The derived service provider can call this method to release user-user information in the **LINECALLINFO** record and cause the next record to be shown.

It is automatically called when a **RTReleaseUserInfo** request completes successfully.

CTSPICallAppearance::RemoveDeviceClass

bool CTSPICallAppearance::RemoveDeviceClass (LPCTSTR *pszClass***);**

pszClass Device class key to remove.

Remarks

This method removes the specified device class information from this object. It will no longer be reported as available to TAPI.

Return Value

TRUE if device class information was located and removed.

CTSPICallAppearance::Secure

virtual LONG Secure (DRV_REQUESTID *dwRequestID***);**

dwRequestID TAPI asynchronous request ID.

Remarks

This method is called in response to an application calling **lineSecure**. The service provider must export the **TSPI_lineSecure** function. The method secures the call from any interruptions or interference that may affect the call's media stream. The class implementation validates that the call is not secure already, verifies that the call is active, and submits a **RTSecureCall** request to the queue.

When the service provider completes the request with a zero result, the call object will set the **LINECALLPARAMFLAGS_SECURE** flag in the **LINECALLINFO.dwCallParamFlags** field.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SendUserUserInfo

virtual LONG SendUserUserInfo (DRV_REQUESTID *dwRequestID***,**
LPCSTR *lpszUserUserInfo***, DWORD** *dwSize***);**

dwRequestID	TAPI asynchronous request ID.
lpszUserUserInfo	Information block to send to the other party
dwSize	Size of the information block to send.

Remarks

This method is called in response to an application calling **lineSendUserUserInfo**. The service provider must export the **TSPI_lineSendUserUserInfo** function. The class implementation validates that the call is active, and submits a **RTSendUserInfo** request to the queue.

If the size of the block to send exceeds the size of a network packet, the service provider is responsible for breaking the block up.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SetAppSpecificData

void SetAppSpecificData(DWORD *dwAppSpecific***);**

dwAppSpecific	Data to set into the structure.

Remarks

This method is used to set the current **LINECALLINFO.dwAppSpecific** field when TAPI calls the **lineSetAppSpecificData** function.

TAPI is notified of the change using a **LINECALLINFOSTATE_APPSPECIFIC** event.

CTSPICallAppearance::SetBearerMode

void SetBearerMode(DWORD *dwBearerMode***);**

dwBearerMode	New bearer mode for the call.

Remarks

This method may be used by the service provider to set the current **LINECALLINFO.dwBearerMode** field. It is used by the class library when a **RTSetCallParams** completes successfully.

TAPI is notified of the change using a **LINECALLINFOSTATE_BEARERMODE** event.

CTSPICallAppearance::SetCallData

virtual LONG SetCallData (DRV_REQUESTID *dwRequestID***,**
LPVOID *lpCallData***, DWORD** *dwSize***);**

dwRequestID	TAPI asynchronous request ID.
lpCallData	Call data buffer to store into the call
dwSize	Size in bytes of the buffer.

Remarks

This method is called in response to an application calling **lineSetCallData**. The service provider must export the **TSPI_lineSetCallData** function. The class implementation validates that the call is active, can support call data of the specified size, and submits a **RTSetCallData** request to the queue.

When the service provider completes the request with a zero result, the call object will set the call data using the method **CTSPICallAppearance::SetCallData**.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SetCallData

void SetCallData (LPVOID *lpvCallData***, DWORD** *dwSize***);**

lpvCallData	Call data to set
dwSize	Size of the call data

Remarks

The service provider can use this method to set the current call data associated with the call object.

It is used by TSP++ when a **RTSetCallData** completes successfully.

The memory used by the call data is allocated internally within the library and freed when either it is replaced by a subsequent call to **SetCallData**, or when the call is destroyed.

TAPI is notified of the change using a **LINECALLINFOSTATE_CALLDATA** event.

CTSPICallAppearance::SetCalledIDInformation

void SetCalledIDInformation (DWORD *dwFlags***,**
LPCTSTR *lpszPartyID* **= NULL, LPCTSTR** *lpszName* **= NULL,**
DWORD *dwCountryCode* **= 0);**
void SetCalledIDInformation (const CALLIDENTIFIER& *ci***);**

dwFlags	**LINECALLPARTYID_xxx** indicating information available.
lpszPartyID	Dialable number to set into call.
lpszName	Name of party.
dwCountryCode	Country code if available.
ci	**CALLIDENTIFIER** block to copy into the called-id record.

Remarks

The service provider can use this method to set the current called id information for the call. The called party corresponds to the originally addressed party.

TAPI is notified of the change using a **LINECALLINFOSTATE_CALLEDID** event.

CTSPICallAppearance::SetCallerIDInformation

void SetCallerIDInformation (DWORD *dwFlags*,
LPCTSTR *lpszPartyID* **= NULL, LPCTSTR** *lpszName* **= NULL,**
DWORD *dwCountryCode* **= 0, DWORD** *dwAddressType* **= *-1L*);**
void SetCallerIDInformation (const CALLIDENTIFIER& *ci*);

dwFlags	**LINECALLPARTYID_xxx** indicating information available.
lpszPartyID	Dialable number to set into call.
lpszName	Name of party.
dwCountryCode	Country code if available.
dwAddressType	Caller's address type (TAPI 3.0 only)
ci	**CALLIDENTIFIER** block to copy into the caller-id record.

Remarks

The service provider can use this method to set the current caller id information for the call. The caller corresponds to the person placing the call.

TAPI is notified of the change using a **LINECALLINFOSTATE_CALLERID** event.

CTSPICallAppearance::SetCallFeatures

void SetCallFeatures (DWORD *dwFeatures*, **bool** *fNotify* = **true);**

dwFeatures	New **LINECALLFEATURE_xx** codes which are available.
fNotify	Whether to notify TAPI about the changes.

Remarks

This method may be used by the service provider to set the current available features for the call appearance. If *fNotify* is set, this method will invoke the **CTSPIAddressInfo:: OnCallFeaturesChanged** method to inform TAPI about the changes.

CTSPICallAppearance::SetCallFeatures2

void SetCallFeatures 2(DWORD *dwFeatures*, **bool** *fNotify* = **true);**

dwFeatures	New **LINECALLFEATURE2_xx** codes which are available.
fNotify	Whether to notify TAPI about the changes.

Remarks

This method may be used by the service provider to set the current available features for the call appearance. If *fNotify* is set, this method will invoke the **CTSPIAddressInfo:: OnCallFeaturesChanged** method to inform TAPI about the changes.

CTSPICallAppearance::SetCallHandle

void SetCallID (HTAPICALL *htCall***);**

htCall TAPI call handle assigned to the call object.

Remarks

This function is used to complete the two-phase construction of the call object. It is invoked by TSP++ to assign the call handle from TAPI into the call object.

CTSPICallAppearance::SetCallID

void SetCallID (DWORD *dwCallID***);**

dwCallID Numeric call identifier to use for this call.

Remarks

In some telephony environments, the switch or service provider may assign a unique identifier to each call. This allows the call to be tracked across transfers, forwards, or other events.

When a call-id is assigned to a call, a *call hub* object is created to represent all calls sharing that call-id. This call hub object can be retrieved from the call using the **GetCallHub** method. If the call-id is changed during the lifetime of the call, the call hub is changed as well.

TAPI is notified through a **LINECALLINFOSTATE_CALLID** event.

CTSPICallAppearance::SetCallOrigin

void SetCallOrigin(DWORD *dwOrigin***);**

dwCallOrigin New **LINECALLORIGIN_xx** value for the call.

Remarks

This changes the call origin of the call. This field is located in the **LINECALLINFO** structure. TAPI is notified through a **LINECALLINFOSTATE_ORIGIN** event.

CTSPICallAppearance::SetCallParameterFlags

void SetCallParameterFlags (DWORD *dwFlags***);**

dwFlags New **LINECALLPARAMFLAG_xxx** values for the call

Remarks

This changes the call parameter flags for the call. These are located in the **LINECALLINFO** structure. TAPI is notified through a **LINECALLINFOSTATE_OTHER** event.

CTSPICallAppearance::SetCallParams

virtual LONG SetCallParams (DRV_REQUESTID *dwRequestID*,
DWORD *dwBearerMode*, **DWORD** *dwMinRate*, **DWORD** *dwMaxRate*,
LPLINEDIALPARAMS *lpDialParams*);

dwRequestID	TAPI asynchronous request ID.
dwBearerMode	Requested bearer mode **LINEBEARERMODE_xxx**.
dwMinRate	Minimum data rate requested.
dwMaxRate	Maximum data rate requested.
lpDialParams	**LINEDIALPARAMS** to set on the call.

Remarks

This method is called in response to an application calling **lineSetCallParams**. The service provider must export the **TSPI_lineSetCallParams** function. The class implementation validates that the call is active and submits a **RTSetCallParams** request to the queue.

When the service provider completes the request with a zero result, the call object will set the new bearer mode and dialing parameter flags into the call object.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SetCallReason

void SetCallReason(DWORD *dwReason*);

dwReason	New **LINECALLREASON_xxx** value for the call

Remarks

This changes the call reason flag for the call. It is located in the **LINECALLINFO** structure. TAPI is notified through a **LINECALLINFOSTATE_REASON** event.

CTSPICallAppearance::SetCallState

void SetCallState(DWORD *dwState*, **DWORD** *dwMode* = **0L**,
DWORD *dwMediaMode* = **0L**, **bool** *fTellTapi* = **TRUE**);

dwState	New call state for the call (**LINECALLSTATE_xxx**).
dwMode	Call state mode (**LINExxxMODE_xxx**).
dwMediaMode	Detected media mode for call (required if first state transition).
fTellTapi	TRUE if TAPI should be notified of the state change.

Remarks

This changes the call state of the given call appearance. When the call is first transitioned out of the **LINECALLSTATE_UNKNOWN** state (which is the initial state), a media mode must be given for the call. The class library changes feature information within the call, address, and line objects when any call changes states. In addition, conferences and attached calls may also react to a call changing state.

TAPI is notified if the **fTellTapi** flag is TRUE using the **LINE_CALLSTATE** event.

CTSPICallAppearance::SetCallTreatment

virtual LONG SetCallTreatment(DRV_REQUESTID *dwRequestID*,
DWORD *dwCallTreatment*);

dwRequestID TAPI asynchronous request ID.
dwCallTreatment New call treatment (**LINECALLTREATMENT_xx**).

Remarks

This method is called in response to an application calling **lineSetCallTreatment**. The service provider must export the **TSPI_lineSetCallTreatment** function. The method sets the sounds a party on a call that is unanswered or on hold hears.

If the call is currently in a state where the call treatment is relevant, then it goes into effect immediately. Otherwise, the treatment will take effect the next time the call enters a relevant state. The class implementation submits a **RTSetCallTreatment** request to the service provider in case any work needs to be done there. If the service provider code completes the request with a zero return code, then the specified call treatment value is stored off in the **LINECALLINFO** record and TAPI is notified of the change.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SetCallTreatment

void SetCallTreatment(DWORD *dwCallTreatment*);

dwCallTreatment New **LINECALLTREATMENT_xxx** value for the call

Remarks

This changes the call treatment flag for the call. It is located in the **LINECALLINFO** structure. TAPI is notified through a **LINECALLINFOSTATE_TREATMENT** event.

CTSPICallAppearance::SetCallType

void SetCallType (int *iCallType*);

iCallType New call type for this call.

Remarks

This changes the call type of the call. The call type is used to identify the type of call appearance this call object represents, a normal call, a conference call, or a consultant call. See **CTSPICallAppearance::GetCallType** for more information.

CTSPICallAppearance::SetChargingInformation

void SetChargingInformation(LPCVOID *lpChargingInformation*,
DWORD *dwSize*);

lpChargingInformation	Charging information to associate to the call.
dwSize	Size of the charging information.

Remarks

This associates the call with ISDN charging information. The data buffer should conform to the ISDN.931 specification.

CTSPICallAppearance::SetConferenceOwner

void SetConferenceOwner(CTSPIConferenceCall* *pCall***);**

pCall	New conference owner for this call.

Remarks

This associates the call with a given active conference on the same line. It is called by the conference call object when calls are inserted and removed from the conference.

CTSPICallAppearance::SetConnectedIDInformation

void SetConnectedIDInformation (DWORD *dwFlags***,**
LPCTSTR *lpszPartyID* **= NULL, LPCTSTR** *lpszName* **= NULL,**
DWORD *dwCountryCode* **= 0, DWORD** *dwAddressType* **=** ***-1L*****);**
void SetConnectedIDInformation (const CALLIDENTIFIER& *ci***);**

dwFlags	**LINECALLPARTYID_xxx** indicating information available.
lpszPartyID	Dialable number to set into call.
lpszName	Name of party.
dwCountryCode	Country code if available.
dwAddressType	Party's address type (TAPI 3.0 only)
ci	**CALLIDENTIFIER** structure to copy connected-id from.

Remarks

The derived service provider may use this method to set the current connected id information for the call. The connected party corresponds to the final party the call connected to, which in most cases will be the called id.

TAPI is notified of the change using a **LINECALLINFOSTATE_CONNECTEDID** event.

CTSPICallAppearance::SetConsultationCall

void SetConsultationCall(CTSPICallAppearance* *pCall***);**

pCall	Call object to set as the consultation for the object.

Remarks

This method establishes a *consultation* relationship between two calls. The passed *pCall* pointer is marked as a consultation call (see **SetCallType**) and then attached to the current object as it's consultation call.

CTSPICallAppearance::SetDataRate

void SetDataRate(DWORD *dwDataRate***);**

dwDataRate New data rate for this call.

Remarks

This method changes the data rate of the call. The data rate starts initially as the current data rate of the address owner. It is not adjusted directly by the class library. The method adjusts the **dwDataRate** field in the **LINECALLINFO** structure and TAPI is notified through a **LINECALLINFOSTATE_RATE** event.

CTSPICallAppearance::SetDestinationCountry

void SetDestinationCountry (DWORD *dwCountryCode***);**

dwCountryCode Country code for call (zero if unknown).

Remarks

This method changes the country code associated with the destination party of the call. It is initially setup on an outgoing call by the method which created the call (**MakeCall**, **Unpark**, etc.). It is not established for incoming calls unless the service provider uses this method to set it. The method adjusts the **dwCountryCode** field in the **LINECALLINFO** structure and TAPI is notified through a **LINECALLINFOSTATE_OTHER** event.

CTSPICallAppearance::SetDialParameters

void SetDialParameters (LINEDIALPARAMS& *dp***);**

dp **LINEDIALPARAMS** to associate with this call appearance.

Remarks

This method changes the dialing parameters associated with the call. Unless these parameters are set by either **MakeCall** or **SetCallParams**, their values will be the same as the defaults used in the **LINEDEVCAPS** of the owning line device. The method adjusts the dialing parameters in the **LINECALLINFO** structure and TAPI is notified through a **LINECALLINFOSTATE_DIALPARAMS** event.

CTSPICallAppearance::SetDigitMonitor

void SetDigitMonitor(DWORD *dwDigitModes***);**

dwDigitModes New digit modes to monitor the device for.

Remarks

This method changes the digit modes that the call is monitoring the media for. The value passed in should be a set of **LINEDIGITMODE_xxx** flags. The method adjusts the **LINECALLINFO. dwMonitorDigitModes** field and notifies TAPI through a **LINECALLINFOSTATE_MONITORMODES** event.

CTSPICallAppearance::SetHiLevelCompatibilityInformation

void SetHiLevelCompatibilityInformation(LPCVOID *lpHLCInformation*, **DWORD** *dwSize*);

lpHLCInformation	Compatibility information to associate to the call.
dwSize	Size of the charging information.

Remarks

This associates the call with ISDN high-level compability information. The data buffer should conform to the ISDN.931 specification.

CTSPICallAppearance::SetLowLevelCompatibilityInformation

void SetLowLevelCompatibilityInformation(LPCVOID *lpLLCInformation*, **DWORD** *dwSize*);

lpLLCInformation	Compatibility information to associate to the call.
dwSize	Size of the charging information.

Remarks

This associates the call with ISDN low-level compability information. The data buffer should conform to the ISDN.931 specification.

CTSPICallAppearance::SetMediaControl

void SetMediaControl (TSPIMEDIACONTROL* *lpMediaControl*);

lpMediaControl	Parameter block associated with request.

Remarks

This method enables and disables control actions on the media stream associated with the specified call. Media control actions can be triggered by the detection of specified digits, media modes, custom tones, and call states. The new specified media controls replace all the ones that were in effect for this line, address, or call prior to this request. The class library removes the previous media control structure and associates itself with the passed media control structure.

Return Value

TAPI error code or zero if the media control structure was associated with the call.

CTSPICallAppearance::SetMediaMode

virtual LONG SetMediaMode (DWORD *dwMediaMode*);

dwMediaMode	New media mode for the call.

Remarks

This method is called in response to an application calling **lineSetMediaMode**. The service provider must export the **TSPI_lineSetMediaMode** function. The method is completely implemented within the class library.

The class library validates the passed media mode, and then changes the current media mode of the call to match the passed media mode. TAPI is notified through a **LINECALLINFOSTATE_MEDIAMODE** event.

Return Value

TAPI error code or zero if the media mode was set into the call.

CTSPICallAppearance::SetMediaMonitor

void SetMediaMonitor(DWORD *dwModes***);**

dwModes	New media modes to monitor the device for.

Remarks

This method changes the media modes that the call is monitoring the media for. The value passed in should be a set of **LINEMEDIAMODE_xxx** flags. The method adjusts the **LINECALLINFO. DwMonitorMediaModes** field and notifies TAPI through a **LINECALLINFOSTATE_MONITORMODES** event.

CTSPICallAppearance::SetQualityOfService

virtual LONG SetQualityOfService (DRV_REQUESTID *dwRequestID***,**
LPVOID *lpSendingFlowSpec***, DWORD** *dwSendingFlowSpecSize***,**
LPVOID *lpReceivingFlowSpec***, DWORD** *dwReceivingFlowSpecSize***);**

dwRequestID	TAPI asynchronous request ID for this request.
lpSendingFlowSpec	Winsock2 **FLOWSPEC** information
dwSendingFlowSpecSize	Size of the buffer in bytes.
lpReceivingFlowSpec	Winsock2 **FLOWSPEC** information
dwRecvgFlowSpecSize	Size of the buffer in bytes

Remarks

This method is called in response to an application calling **lineSetCallQualityOfService**. The service provider must export the **TSPI_lineSetCallQualityOfService** function.

One of the new features in TAPI 2.x is the ability to request, negotiate, renegotiate, and receive indications of Quality of Service (QOS) parameters on incoming and outgoing calls. QOS information is exchanged between applications and service providers in **FLOWSPEC** structures that are defined in Windows Sockets 2.0.

The class library verifies that the call is not **Idle**, and submits a **RTSetQualityOfService** request to the line queue.

If the service provider completes the request successfully, then the new QOS information is set into the **LINECALLINFO** structure using the method **CTSPICallInfo::SetQualityOfService**.

Return Value

TAPI error code or the asynchronous request ID if the request was added to the queue.

CTSPICallAppearance::SetQualityOfService

void SetQualityOfService (LPVOID *lpSendingFlowSpec***, DWORD** *dwSendSize***,**
LPVOID *lpReceivingFlowSpec***, DWORD** *dwReceiveSize***);**

lpSendingFlowSpec	**FLOWSPEC** structure for sending information.
dwSendSize	Size of the sending structure.
lpReceivingFlowSpec	**FLOWSPEC** structure for receiving information.
dwReceiveSize	Size of the receiving structure

Remarks

This method changes current Quality of Service (QOS) information associated with a call appearance. It should be called when the QOS information is renegotiated on a call by the hardware or the other side of the connection. The method deletes the existing QOS information, and copies the new information into the call appearance. TAPI is then notified through a **LINECALLINFO_QOS** event.

CTSPICallAppearance::SetRedirectingIDInformation

void SetRedirectingIDInformation (DWORD *dwFlags***,**
LPCTSTR *lpszPartyID* **= NULL, LPCTSTR** *lpszName* **= NULL,**
DWORD *dwCountryCode* **= 0, DWORD** *dwAddressType* **=** ***-1L*****);**
void SetRedirectingIDInformation (const CALLIDENTIFIER& *ci***);**

dwFlags	**LINECALLPARTYID_xxx** indicating information available.
lpszPartyID	Dialable number to set into call.
lpszName	Name of party.
dwCountryCode	Country code if available.
dwAddressType	Party's address type (TAPI 3.0 only)
ci	**CALLIDENTIFIER** structure to copy redirecting information from.

Remarks

The service provider can use this method to set the current redirecting id information for the call. The redirecting party corresponds to the party that directed the call to its destination.

TAPI is notified of the change using a **LINECALLINFOSTATE_REDIRECTINGID** event.

CTSPICallAppearance::SetRedirectionIDInformation

void SetRedirectionIDInformation (DWORD *dwFlags***,**
LPCTSTR *lpszPartyID* **= NULL, LPCTSTR** *lpszName* **= NULL,**
DWORD *dwCountryCode* **= 0, DWORD** *dwAddressType* **=** ***-1L*****);**
void SetRedirectionIDInformation (const CALLIDENTIFIER& *ci***);**

dwFlags	**LINECALLPARTYID_xxx** indicating information available.
lpszPartyID	Dialable number to set into call.
lpszName	Name of party.
dwCountryCode	Country code if available.
dwAddressType	Party's address type (TAPI 3.0 only)

ci	**CALLIDENTIFIER** structure to copy redirection information from.

Remarks

The service provider can use this method to set the current redirected id information for the call. The redirected party corresponds to the party where the call diverted to as a result of a forward, transfer, etc.

TAPI is notified of the change using a **LINECALLINFOSTATE_REDIRECTIONID** event.

CTSPICallAppearance::SetRelatedCallID

void SetRelatedCallID (DWORD *dwCallID***);**

dwCallID	Call ID to which this call is *related*.

Remarks

This method is used to associate the current call with another call.

The related call-id field is currently used by TSP++ to associate consultation calls created on behalf of another call.

TAPI is notified of the change using a **LINECALLINFOSTATE_RELATEDCALLID** event.

CTSPICallAppearance::SetTerminalModes

void SetTerminalModes (int *iTerminalID***, DWORD** *dwTerminalModes***, bool** *fRouteToTerminal***);**

iTerminalID	Terminal identifier to adjust (0 to **GetTerminalCount**).
dwTerminalModes	Terminal mode(s) (**LINETERMMODE_xxx**) to adjust.
f*RouteToTerminal*	Whether the terminal is added or removed from modes.

Remarks

This is the method that is called when the terminal routing is adjusted by either TAPI or through the hardware. This stores or removes the specified terminal from the terminal modes given.

TAPI is notified about the change through a **LINECALLINFO_TERMINAL** event.

CTSPIAddressInfo::SetTrunkID

void SetTrunkID (DWORD *dwTrunkID***);**

dwTrunkID	New trunk ID for the call.

Remarks

This method changes the current external trunk that is associated with the call. The service provider should set the trunk value as soon as it is received from the hardware.

TAPI is notified about the change through a **LINECALLINFO_TRUNK** event.

CTSPICallAppearance::SwapHold

virtual LONG SwapHold(DRV_REQUESTID *dwRequestID***,**
CTSPICallAppearance* *pCall***);**

dwRequestID	Asynchronous request ID for this request.
pCall	Call appearance to swap with.

Remarks

This method is called in response to an application calling **lineSwapHold**. The **TSPI_lineSwapHold** function must be exported from the provider. The default implementation of the library is to verify both calls to insure the proper call states. Then to create and insert a **RTSwapHold** request into the queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::Unhold

virtual LONG Unhold (DRV_REQUESTID *dwRequestID***);**

dwRequestID	Asynchronous request ID for this request.

Remarks

This method is called in response to an application calling **lineUnhold**. The **TSPI_lineUnhold** function must be exported from the provider. The default implementation of the library is to verify that the call is **onHold**, then to create and insert a **RTUnhold** request into the queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallAppearance::Unpark

virtual LONG Unpark (DRV_REQUESTID *dwRequestID***,**
TDialStringArray* *parrAddresses***);**

dwRequestID	Asynchronous request ID for this request.
parrAddresses	Address to retrieve the call from.

Remarks

This method is called in response to an application calling **lineUnpark**. The **TSPI_lineUnpark** function must be exported from the provider. The class library verifies that an address is given in the array, and submits a **RTUnpark** request into the line queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPIConferenceCall

A **CTSPIConferenceCall** object represents a conference call in the TSP++ library. This object is derived from the **CTSPICallAppearance** object and therefore behaves in the same fashion as any other call appearance. The main difference is that the conference call object is a placeholder for a series of call appearances that comprise the conference. As each call is added to the conference, this object will add the call appearance to a list of active conferenced calls. When this object is deleted or the state is changed to **Idle**, it will automatically de-associate all the call appearances that were previously part of the conference. As each conferenced call changes state, the conference call object gets notified and performs various actions.

Conferences

Conference calls are call appearances that include more than two parties simultaneously. A conference call can be established in a number of ways, depending on hardware device capabilities. The capabilities of the line device may limit the number of parties conferenced into a single call and whether or not a conference must start with a connected call. It can also affect how the conference is broken down when it ends. The standard methods supported by TAPI to create a conference are:

- A conference call may need to be established with **lineSetupConference** without an existing two-party call. This returns a handle for the conference call and allocates a consultation call. After a period of consultation, the consultation call can be added with **lineAddToConference**.
- A conference call may begin as a regular two-party call, inbound or outbound. Calling **lineSetupConference** establishes the conference call. This operation takes the original two-party call as input, allocates a conference call, connects the original call to the conference, and allocates a consultation call whose handle is returned to the application. The original call is placed into the **onHoldPendingConf** state, and the new consultation call should be in the **Dialtone** state. The **lineDial** method can then be used on the consultation call to establish a connection to the next party to be added. Once connected to another party, the **lineAddToConference** method adds the new call to the conference.
- A three-way conference call can be established by resolving a transfer request into three-way conference. In this scenario, a two-party call is established as either an inbound or outbound call. Next the call is placed on transfer hold with the function **lineSetupTransfer** which returns a consultation call handle. After a period of consultation, the application may have the option to resolve the transfer setup by selecting the three-way conference option which conferences all three

parties together in a conference call with **lineCompleteTransfer** with the **conference** option. Under this option, a conference call handle representing the conference call is allocated and returned to TAPI?

To add additional parties to an existing conference call, TAPI will invoke the **linePrepareAddToConference** method.. When calling this function, the application supplies the handle of an existing conference call. The method allocates a consultation call that can later be added to the conference call and returns a consultation call handle to TAPI. This conference call is then transitioned to the **onHoldPendingConf** state. Once the consultation call exists, it can be added to the existing conference call with **lineAddToConference**. In some cases, it may be allowable to take any existing call and conference it into an existing conference by placing the call on hold and using **lineAddToConference**.

Once a call becomes a member of a conference call, the member's call state reverts to **Conferenced.** The state of the conference call typically becomes **Connected**. The call handle to the conference call and all the added parties remain valid as individual calls. As each member disconnects from the conference by hanging up, the appropriate call state messages are sent by the library to inform TAPI of this fact. TAPI may also direct for a call to be removed from the conference via the **lineRemoveFromConference** method. The **LINEDEVCAPS** in the **CTSPILineConnection** object describe how removal from a conference may occur.

Other notes about conferencing with TSP++

Since conference calls are generally hardware specific implementations, not much support is provided in the library for actual management of the conference other than the automatic tying together of all the calls (call association). When a **Drop** request is issued for a conference call handle, the derived service provider will probably have to perform some work with the calls within the conference. The **GetConferenceCount** and **GetConferenceCall** methods allow access to each call which transitions into the **Conference** state.

Constructor

CTSPIConferenceCall	Constructor for the conference object

Operations - Public Members

AddToConference	This method manually builds a conference by adding a single call.
GetConference	Return a specific **CTSPICallAppearance** which has been conferenced in and is in the **Conferenced** state.
GetConferenceCount	Return the count of conferenced calls within this conference.
RemoveConferenceCall	Remove a call appearance from the conference.

Operations - Protected Members

CanRemoveFromConference	Returns a TRUE/FALSE result indicating whether the specified call may be removed from the owner conference.
IsCallInConference	Return a TRUE/FALSE result indicating whether the specified call appearance exists in our conference call list.

Overridables - Protected Members

OnCallStatusChange	Overridden from the **CTSPICallAppearance** class in order to break-down a conference call in the proper order. When the conference call goes idle, any existing calls attached to the conference will transition to the idle state automatically *before* TAPI is notified about the conference call going idle.
OnConsultantCallIdle	This is called when the consultation call attached to this call is transitioned to the idle state. This by default does nothing, but allows the derived class to perform whatever action the switch normally takes (i.e. move back to the connected state, establish a new consultation call, etc.)
OnRelatedCallStateChange	Overridden from the **CTSPICallAppearance** class to receive notifications when related calls change call state. This method is responsible to addition and deletion of calls from the conference.
OnRequestComplete	This method is used to monitor requests which complete and are associated with this conference.

Overridables - TAPI Members

AddToConference	Add an existing consultation call to an existing conference. Submits an asynchronous request. This is called by the **lineAddToConference** method.
PrepareAddToConference	Prepare to add a new call to the conference. Submits an asynchronous request. This is called by the **linePrepareAddToConference** method.
RemoveFromConference	Removes a call appearance that is conferenced in. This is call by the **lineRemoveFromConference** method.

CTSPIConferenceCall::AddToConference

virtual LONG AddToConference (DRV_REQUESTID *dwRequestID***,**
CTSPICallAppearance* *pCall***);**

dwRequestID	Asynchronous request ID for this request.
pCall	Call to add to the conference

Remarks

This method is called in response to an application calling **lineAddToConference**. The **TSPI_lineAddToConference** function must be exported from the provider.

TSP++ checks the call which is being added to the conference to make sure that it is either

1. The consultant call created by a call to **TSPI_linePrepareAddToConference**.
2. The provider can handle addition of *any* call to a conference (the **LINEADDRESSCAPS.dwAddrCapFlags** has the **LINEADDRCAPFLAGS_CONFERENCEMAKE** flag).

Next, the library checks to see if the call is on the same address – if not, the **LINEDEVCAPS.dwDevCapFlags** must have the **LINEDEVCAPFLAGS_CROSSADDRCONF**flag set.

The call-state of the conference and call to add are checked, the **dwRelatedCallID** field of the consultation call is set to the conference call, and a **RTAddToConference** request is submitted to the line.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPIConferenceCall::AddToConference

void AddToConference(CTSPICallAppearance* *pCall***);**

pCall	Call to add to the conference.

Remarks

This method adds the specified call directly to the conference. It may be used by the service provider to build a conference directly off the hardware responses – specifically when a conference is being built through an external source, such as a phone or another extension.

CTSPIConferenceCall::CanRemoveFromConference

Protected

bool CanRemoveFromConference(CTSPICallAppearance* *pCall***) const;**

pCall	Call to remove from the conference.

Remarks

This method checks to determine if the specified call may be removed from the conference. It checks to see how a conference call must be disassembled according to the rules set into the **LINEADDRESSCAPS.dwRemoveFromConfCaps** field.

Return Value

TRUE if the call may be removed from the conference now. FALSE otherwise.

CTSPIConferenceCall:: CTSPIConferenceCall

CTSPIConferenceCall();

Remarks

This is the constructor for the conference call object.

CTSPIConferenceCall:: GetConferenceCall

CTSPICallAppearance* GetConferenceCall(unsigned int *iPos***);**

iPos Position within the conference array (0 to count).

Remarks

This method retrieves a call appearance that is part of the given conference call. Conference calls are stored in an array managed by the conference owner. The total count of members within the conference may be retrieved through the **CTSPIConferenceCall::GetConferenceCount** method.

Return Value

Pointer to the **CTSPICallAppearance** which is at the specified position in the conference array. NULL if no call is at that position.

CTSPIConferenceCall:: GetConferenceCount

unsigned int GetConferenceCount() const;

Remarks

This method retrieves the count of calls that are members of the conference.

Return Value

Total number of calls that are part of the conference.

CTSPIConferenceCall:: IsCallInConference

Protected

bool IsCallInConference(CTSPICallAppearance* *pCall***) const;**

pCall Call object in question.

Remarks

This method walks the conference array and returns whether or not the specified call is part of the conference.

Return Value

TRUE if the call is currently in the conference. FALSE otherwise.

CTSPIConferenceCall::OnRequestComplete

Protected

virtual void OnRequestComplete (CTSPIRequest* *pReq*, **LONG** *lResult*);

pReq	Request which has completed
lResult	Final return code for the request.

Remarks

This method is called when a request is completed on the owner address/line. It gives the call object an opportunity to cleanup any data or information associated with the request.

The default implementation of the class library manages several requests:

RTRemoveFromConference If the request completes successfully, the specified call is removed from the conference if it wasn't already by the service provider making the call **Idle**.

CTSPIConferenceCall::PrepareAddToConference

virtual LONG PrepareAddToConference(DRV_REQUESTID *dwRequestID*,
HTAPICALL *htConsultCall*, **LPHDRVCALL** *lphdConsultCall*,
LPLINECALLPARAMS *lpCallParams*);

dwRequestID	Asynchronous request ID for this request.
htConsultCall	TAPI opaque handle to created consultation call
lphdConsultCall	TSP returning call handle to consultation call.
lpCallParams	**LINECALLPARAMS** for the newly created consultation call.

Remarks

This method is called in response to an application calling **linePrepareAddToConference**. The **TSPI_linePrepareAddToConference** function must be exported from the provider. The class library validates the conference parameters, validates the total count within the conference, creates the consultation call and associates it with the conference, and finally, submits a **RTPrepareAddToConference** request to the line queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPIConferenceCall::RemoveConferenceCall

void RemoveConferenceCall(CTSPICallAppearance* *pCall*,
bool *fForceBreakdown*=**true);**

pCall	Call object to remove from the conference
fForceBreakdown	This determines whether the conference will transition to the **Idle** state if this was the final call. If this is set to false, then the conference will stay in existence.

Remarks

This method removes the specified call from the conference, possibly destroying the conference call if there is only a single party left once the removal is complete.

CTSPIConferenceCall::RemoveFromConference

virtual LONG RemoveFromConference(DRV_REQUESTID *dwRequestID*,
CTSPICallAppearance* *pCall*);

dwRequestID	Asynchronous request ID for this request.
pCall	Call to remove from the conference

Remarks

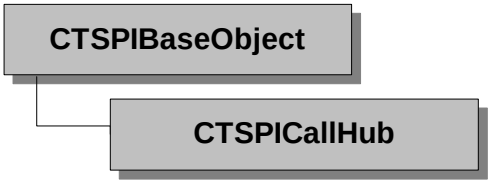
This method is called in response to an application calling **lineRemoveFromConference**. The **TSPI_lineRemoveFromConference** function must be exported from the provider.

The class library validates the conference parameters, makes sure the call may be removed by using the **CTSPIConferenceCall::CanRemoveFromConference** method, and submits a **RTRemoveFromConference** request into the line queue.

Return Value

TAPI error code or the asynchronous request ID if the request was queued.

CTSPICallHub



When a full PBX is modeled rather than just a few lines, a new problem in tracking calls by call-id is introduced. TAPI presents a call object for each line object – even when the call is actually in-switch. This presents a situation where two call objects represent the same physical conversation on the switch. The TSP will now see both sides of a given call. In a first-party environment, the call-id is generally always associated with only one call, in the third-party system, two or more calls can exist that share a single call-id.
To facilitate this process, the call objects have an internal pointer to a *call hub* owner that maps all the calls that share that specific call-id (and are therefore mirrors of the same call on the switch). This **CTSPICallHub** object reaches across lines and makes it easy to find the other side of a conversation, which is absolutely necessary, when calls are being transferred. The call hub object can be located using the call object's **GetCallHub** method.

Global call-id lookups

A map of call-ids to call hub objects is maintained at the device level (**CTSPIDevice**). This allows the service provider to quickly map a call-id on the switch to all the calls that have been created for it. This is done through the **FindCallHub** method of the device object.
The only call-id which does *not* have a hub object is zero. If a call is assigned the call-id of zero, no call hub is stored in the hub pointer. This may be a problem on some switches that allow for a zero call-id (not many do), and JulMar suggests that the derived provider increment the call-id by one if the switch uses zero as a valid call-id.

Lifetime of the hub object

In a telephony environment, call-ids are very dynamic. This makes the hub object dynamic as well. It is created and destroyed as call-ids are created and destroyed. When the first call associated with a call-id is created, a call-hub is created to track the call-id. When the last call for that call-id is deleted, the call hub is deleted with it.
Calls which change call-id are automatically moved in and out of the appropriate hub objects by TSP++.
Unlike other telephony objects in TSP++, call hubs cannot be overridden.

Constructor and Destructor

CTSPICallHub	Constructor for the call hub object.
~CTSPICallHub	Destructor for the call hub object.

Operations - Public Members

GetHubCount	Returns the count of calls in the hub.
GetCall	Returns a single call in the hub by index.
GetShadowCall	Finds the other side of a two-party call.
IsCallInHub	Returns whether the given call is in the hub.
AddToHub	Adds a call to the hub.
RemoveFromHub	Removes a call from the hub, deletes the hub if this is the last call object.
OnCallStateChange	Notification about call state changes from a call in the hub – forwards the event to all the other calls using the **OnShadowCallStateChange** method of the call object.

CTSPICallHub::AddToHub

void AddToHub(CTSPICallAppearance* *pCall***);**

pCall Call object to add to the hub

Remarks

This method adds the given call object to the hub. The call should have the same call-id that the hub was created for.

CTSPICallHub::CTSPICallHub

CTSPICallHub(DWORD *dwCallID***);**

dwCallID Call-id this hub is managing

Remarks

This is the constructor for the call hub object.

CTSPICallHub::~CTSPICallHub

~CTSPICallHub();

Remarks

This is the destructor for the call hub object.

CTSPICallHub::GetCall

CTSPICallAppearance* GetCall(unsigned int *iPos***) const;**

iPos Array position to retrieve the call from.

Remarks

This method retrieves a call object from the call hub based on the index array position. This must be between zero and **GetHubCount**.

Return Value

Pointer to the retrieved call object, NULL if no call exists at that position.

CTSPICallHub::GetHubCount

unsigned int GetHubCount() const;

Remarks

This method returns the number of calls which are part of this hub.

Return Value

Numeric number of calls, should always be greater than zero.

CTSPICallHub::GetShadowCall

CTSPICallAppearance* GetShadowCall(
const CTSPICallAppearance* *pCall*) **const;**

pCall	Call to locate the shadow for

Remarks

This method retrieves a call object from the hub which is determined to be the other side of the passed call.

This function works if there are only two calls in the hub, any more or less and the function returns NULL.

Return Value

Pointer to the retrieved call object, NULL if no call shadow can be determined.

CTSPICallHub::IsCallInHub

bool IsCallInHub(const CTSPICallAppearance* *pCall*) **const;**

pCall	Call to locate in the hub

Remarks

This method returns whether the passed call object is part of this hub.

Return Value

TRUE if the call is part of this hub, FALSE if not.

CTSPICallHub::OnCallStateChange

void OnCallStateChange(CTSPICallAppearance* *pCall*,
DWORD *dwState*, **DWORD** *dwCurrState*) **const;**

pCall	Call which has changed call-state
dwState	New state of the given call.
dwCurrState	Previous state of the call.

Remarks

This method is used to notify other calls in a hub that a shadow has changed status information. It calls the **OnShadowCallStateChange** method for all of the calls (excluding the passed call) in the hub.

CTSPICallHub::RemoveFromHub

void RemoveFromHub(const CTSPICallAppearance* *pCall*);

pCall	Call to remove from the hub

Remarks

This method is used to remove a given call from the hub. If this is the last call in the hub, the hub is destroyed automatically.

CMSPDriver

In TAPI 2.x and below, media is managed using the multimedia APIs; specifically, the WAV functions. In order to play or record media streams, a WAV driver must be developed to control access to the hardware.
TAPI 3.0 introduced the concept of a *Media Stream Provider* (MSP). This provider is a different type of device driver which conforms to the DirectShow media API. Some new TSPI functions were added to integrate the MSP interfaces into the TAPI system. These interfaces allow the TAPI server to identify the appropriate MSP for a given set of line devices.
The **CMSPDriver** object is used to encapsulate the interface between an MSP and a TSP driver. It does not in any way help further the development of the MSP – that is a topic beyond the scope of TSP++. *For more information on developing MSPs, see the MSDN SDK on Media Service Providers.*

Lifetime of an MSP driver object

The MSP driver object is created when a new MSP instance is created by the TAPI Server. This is accomplished through the **TSPI_lineCreateMSPInstance**. The **CTSPILineConnection::CreateMSPInstance** handler allocates a new **CMSPDriver** object each time it executes. When the MSP interface is closed through **TSPI_lineCloseMSPInstance**, the handler **CTSPILineConnecton::CloseMSPInstance** deallocates and destroys the associated **CMSPDriver** object. The lifetime is completely controlled by TSP++ and, in general, will not need to be changed by a derived provider. Unlike other telephony objects in TSP++, MSP driver objects cannot be overridden.

Constructor and Destructor

CMSPDriver	Constructor for the MSP object.
~CMSPDriver	Destructor for the MSP object.

Operations - Public Members

GetAddressOwner	Returns the address associated with the open MSP interface.
GetLineOwner	Returns the line associated with the open MSP interface.
GetTAPIHandle	Returns the **HTAPIMSPLINE** handle which represents TAPIs opaque version of the object.
SendData	Sends a block of data to the media service provider which is connected through this object. The format of the data is specific to the MSP.

CMSPDriver::CMSPDriver

CMSPDriver(CTSPIAddressInfo* *pOwner***, HTAPIMSPLINE** *htHandle***);**

pOwner	Line which owns this MSP driver interface
htHandle	Opaque TAPI representation of the MSP connection

Remarks

Constructor for the **CMSPDriver** object

CMSPDriver::~CMSPDriver

virtual ~CMSPDriver();

Remarks

Destructor for the **CMSPDriver** object

CMSPDriver::GetAddressOwner

CTSPIAddressInfo* GetAddressOwner() const;

Remarks

This function returns the address owner which was established when the MSP driver object was created and the relationship to the media service provider was setup.

Return Value

A pointer to the **CTSPIAddressInfo** object which this MSP interface is connected to.

CMSPDriver::GetLineOwner

CTSPILineConnection* GetLineOwner() const;

Remarks

This function returns the line owner which was established when the MSP driver object was created and the relationship to the media service provider was setup.

Return Value

A pointer to the **CTSPILineConnection** object which this MSP interface is connected to.

CMSPDriver::GetTAPIHandle

HTAPIMSPLINE GetTapiHandle() const;

Remarks

This function returns the opaque TAPI handle which represents the connection to the media service provider.

Return Value

The **HTAPIMSPLINE** handle which TAPI uses to refer to this connection.

CMSPDriver::SendData

void SendData(CTSPICallAppearance* *pCall***, LPVOID** *lpData***,**
DWORD *dwSize***);**

pCall	Call which is sending the data (can be NULL)
lpData	Data block to send
dwSize	Size of the above data block to marshal over.

Remarks

This function sends a block of data to the media service provider which this object represents. The response will be received by either **CTSPILineConnection::ReceiveMSPData** or **CTSPICallAppearance::ReceiveMSPData** depending on whether a call is targeted by the MSP.

RTAccept

This is generated in response to the **TSPI_lineAccept** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, the call object will be transitioned to the **LINECALLSTATE_ACCEPTED** state *if* it is still in the **LINECALLSTATE_OFFERING** state and the **ACCEPTED** state is a valid callstate in the **LINEADDRESSCAPS.dwCallStates** field.

Failure

None

RTAccept::GetSize

DWORD GetSize() const;

Remarks

This method returns the size of the user-user information block which may be zero.

Return Value

Returns the size of the user-user information block.

RTAccept::GetUserUserInfo

LPCSTR GetUserUserInfo() const;

Remarks

Returns the user-user information buffer which is to be sent with this accept request. The size was validated against the **LINEDEVCAPS. dwUUIAcceptSize** field.

Return Value

A NULL terminated string of user-user information to deliver with the accept request to the switch.

RTAddToConference

This is generated in response to the **TSPI_lineAddToConference** function.

Automatic Handling by TSP++

Success

This request is not directly handled by TSP++ but if the call being added to the conference is transitioned to the **LINECALLSTATE_CONFERENCED** state then the call will be automatically added to the conference it is attached to.

Failure

None

RTAddToConference::GetConferenceCall

CTSPIConferenceCall* GetConferenceCall() const;

Return Value

Returns the conference call object that this request relates to.

RTAddToConference::GetConsultationCall

CTSPICallAppearance* GetConsultationCall() const;

Return Value

Returns the consultation call which is being added to the conference.

RTAgentSpecific

This is generated in response to the **lineAgentSpecific** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTAgentSpecificGetBuffer

LPVOID GetBuffer();

Return Value

Returns the buffer passed with the request. This may be NULL depending on the extension being invoked.

RTAgentSpecific::GetBufferSize

DWORD GetBufferSize() const;

Return Value

Returns the buffer size for the passed data buffer.

RTAgentSpecific::GetExtensionID

DWORD GetExtensionID() const;

Return Value

Returns the extension identifier for the agent extension being invoked.

RTAnswer

This is generated in response to the **TSPI_lineAnswer** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTAnswer::GetSize

DWORD GetSize() const;

Return Value

Returns the size of the user-user information block.

RTAnswer::GetUserUserInfo

LPCSTR GetUserUserInfo() const;

Return Value

Returns the user to user buffer which is to be sent with this answer request. The size was validated against the **LINEDEVCAPS. dwUUIAnswerSize** field.

RTBlindTransfer

This is generated in response to the **TSPI_lineBlindTransfer** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTBlindTransfer::DialArray

TDialStringArray* DialArray();

Return Value

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTBlindTransfer::GetCount

int GetCount() const;

Return Value
Returns the count of dialable addresses passed with this request.

RTBlindTransfer::GetCountryCode

DWORD GetCountryCode() const;

Return Value
Returns the country code specified for this request. This will normally be zero.

RTBlindTransfer::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

Return Value
This retrieves a specific **DIALINFO** record which was passed to the request from TAPI.

RTCompleteCall

This is generated in response to the **TSPI_lineCompleteCall** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTCompleteCall::GetCompletionID

```
DWORD GetCompletionID() const;
```

Return Value

This method returns the unique identifier assigned to this completion request by TAPI.

RTCompleteCall::GetCompletionMode

```
DWORD GetCompletionMode() const;
```

Return Value
This returns the completion mode (**LINECALLCOMPLMODE_xxx**) asked for by the application. This was validated against the **LINEADDRESSCAPS.dwCallCompletionModes** value.

RTCompleteCall::GetMessageID

```
DWORD GetMessageID() const;
```

Return Value
Message identifier to forward to the station. This is a value from zero to **LINEADDRESSCAPS.dwNumCompletionMessages**.

RTCompleteCall::GetNumericIdentifier

```
DWORD GetNumericIdentifier() const;
```

Return Value

This returns the numeric identifier established by the derived service provider when it issued the call completion request to the switch. This should have been done using the **SetIdentifier** method of this object.

RTCompleteCall::GetStringIdentifier

LPCTSTR GetStringIdentifier() const;

Return Value
This returns the text identifier established by the derived service provider when it issued the call completion request to the switch. This should have been done using the **SetIdentifier** method of this object.

RTCompleteCall::SetIdentifier

void SetIdentifier(LPCTSTR *pszSwitchInfo***, DWORD** *dwSwitchInfo***=0);**
void SetIdentifier(DWORD *dwSwitchInfo***);**

pszSwitchInfo	A textual identifier for the completion request
dwSwitchInfo	A numeric identifier for the completion request

Remarks
This function should be used by the service provider when the complete call request is sent to the PBX. In general this would be an extension which will appear on the display, or an id number generated by the switch, etc. Basically it is something to positively identify an incoming call as a call completion request.

RTCompleteTransfer

This is generated in response to the **TSPI_lineCompleteTransfer** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, then the consultation call relationship between the two presented calls is removed.

Failure

If the request is completed with a non-zero return code, the following steps are taken by TSP++ to cleanup the object:

1. The conference call (if any was created for this request) is deallocated if it has not yet been shown to TAPI.
2. The conference owner of the consultation call and call to transfer are both set to NULL.

RTCompleteTransfer::GetCallToTransfer

CTSPICallAppearance* GetCallToTransfer() const;

Return Value

This returns the call which is to be transferred.

RTCompleteTransfer::GetConferenceCall

CTSPIConferenceCall* GetConferenceCall() const;

Return Value

This returns the created conference call for this transfer event. If the transfer is not creating a conference this will be NULL.

RTCompleteTransfer::GetConsultationCall

CTSPICallAppearance* GetConsultationCall() const;

Return Value

This returns the consultation call which is the target of the transferred call.

RTCompleteTransfer::GetTransferMode

DWORD GetTransferMode() const;

Return Value

This returns the transfer mode to complete (**LINETRANSFERMODE_xxx**). It was validated against the **LINEADDRESSCAPS.dwTransferModes** field.

RTDial

This is generated in response to the **TSPI_lineDial** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTDial::DialArray

TDialStringArray* DialArray();

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTDial::GetCount

int GetCount() const;

Returns the count of dialable addresses passed with this request.

RTDial::GetCountryCode

DWORD GetCountryCode() const;

Returns the country code specified for this request. This will normally be zero.

RTDial::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

This retrieves a specific **DIALINFO** record that was passed to the request from TAPI.

RTDropCall

This is generated in response to the **TSPI_lineDrop** function.

Automatic Handling by TSP++

Success

If the request completes successfully then all pending requests for the call are thrown out of the request queue (and returned to TAPI as **LINEERR_OPERATIONFAILED**). If the call is not really being dropped (the **IgnoringDrop** method was called by the service provider) then the **_IsDropped** flag is cleared so that if the call ever is shown to TAPI in the future it may be dropped.

Failure

If the request is completed with a non-zero return code the **_IsDropped** flag is cleared from the call flags so that future attempts to call **lineDrop** will not be refused by TSP++.

RTDropCall::GetSize

DWORD GetSize() const;

Return Value

Returns the size of the user-user information block.

RTDropCall::GetUserUserInfo

LPCSTR GetUserUserInfo() const;

Return Value

Returns the user-user information buffer which is to be sent with this accept request. The size was validated against the line device capabilities.

RTDropCall::IgnoringDrop

void IgnoringDrop(bool *fIsIgnoring* **= true);**

fIsIgnoring	Whether TSP++ should ignore this drop request.

Remarks

This function is used to tell TSP++ to ignore the drop completion. Normally when a drop request is completed with a zero return code, TSP++ begins to initiate call-cleanup operations against a call. This tells TSP++ that the drop is *not* going to happen even though the request might complete successfully.
It should be used when TAPI requests a drop due to the line closing but the TSP doesn't necessarily want to drop the calls.

RTDropCall::IsIgnoringDrop

bool IsIgnoringDrop() const;

Return Value

This returns the value of the internal field set by the **IgnoringDrop** method. It is by default set to false unless changed by the service provider worker code.

RTDropCall::IsImplicitDropFromLineClose

bool IsImplicitDropFromLineClose() const;

Return Value

This returns TRUE if the drop request is coming from TAPISRV because all applications have closed their line handle references to the owner line. This means that no application specifically requested a drop.

RTForward

This is generated in response to the **TSPI_lineForward** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, the following actions are taken by TSP++ for each address object the forward event affected:

1. The **LINEADDRESSSTATUS. dwNumRingsNoAnswer** field is updated with the new value from the request.
2. The **TSPIFORWARDINFO** structures are copied into the address array for reporting through the **TSPI_lineGetAddressStatus** function.
3. TAPI is notified that the forwarding information for the address has changed.

Failure

If the forward request is completed with a non-zero return code, then the created consultation call (if any) is deallocated if it was never shown to TAPI (i.e. never sent a call state event).

RTForward::GetCallParameters

```
LPLINECALLPARAMS GetCallParameters();
```

Return Value

This returns the calling parameters for the consultation call which may have been created as part of this request.

RTForward::GetConsultationCall

```
CTSPICallAppearance* GetConsultationCall() const;
```

Return Value

This returns the consultation call that was created as a result of the fowarding request. On some PBX systems, initiating the forward action causes a call to be created in the dialtone state. This will be NULL unless the **LINEADDRESSCAPS.dwAddrCapsFlags** field has the **LINEADDRCAPFLAGS_FWDCONSULT** bit set in it.

RTForward::GetForwardingAddressCount

```
unsigned int GetForwardingAddressCount() const;
```

Return Value

This returns the count of the **TSPIFORWARDINFO** structures associated with this request.

RTForward::GetForwardingArray

TForwardInfoArray* GetForwardingArray();

Return Value

This returns the array of **TSPIFORWARDINFO** structures which detail the forwarding instructions. It is preferable to use the **GetForwardingAddressCount** and **GetForwardingInfo** methods to access this array.

RTForward::GetForwardingInfo

TSPIFORWARDINFO* GetForwardingInfo(unsigned int *iIndex***);**

iIndex The index of the forwardinfo structure to retrieve.

Return Value

This allows access to each of the **TSPIFORWARDINFO** structures which detail the forwarding instructions.

RTForward::GetNoAnswerRingCount

DWORD GetNoAnswerRingCount() const;

Return Value

This returns the number of rings before a call should be forwarded as per the instructions in the **TSPIFORWARDINFO** structures.

RTGenerateDigits

This is generated in response to the **TSPI_lineGenerateDigits** function.
Note that the **lineGenerateDigits** function is *not* an asynchronous event. Therefore the notification for success or failure is accomplished using an event callback through the **LINE_GENERATE** line device event.

Automatic Handling by TSP++

Success

If the request is completed successfully, TAPI is notified about the digit completion through a **LINEGENERATETERM_DONE** event code.

Failure

If the request is completed with a non-zero return code, TAPI is notified that the digit generation was canceled through a **LINEGENERATETERM_CANCEL** event code.

RTGenerateDigits::GetDigitMode

```
DWORD GetDigitMode() const;
```

Return Value

This returns the **LINEDIGITMODE_xxx** constant of the type of digit generation that is to occur on the call. It was validated by TSP++ using the **LINEDEVCAPS.dwGenerateDigitModes** field.

RTGenerateDigits::GetDigits

```
TString& GetDigits();
```

Return Value

This is the string of digits to generate. It will always be composed of (0-9, A-D, #, *).

RTGenerateDigits::GetDuration

```
DWORD GetDuration() const;
```

Return Value

This is the duration to generate the digits for. This value was adjusted by TSP++ to make sure it is within the minimum/maximum values presented in the **LINEDEVCAPS** structure.

RTGenerateDigits::GetIdentifier

DWORD GetIdentifier() const;

Return Value

This is the unique identifier for this generation request assigned by TAPI.

RTGenerateTone

This is generated in response to the **TSPI_lineGenerateTone** function.
Note that the **lineGenerateTone** function is *not* an asynchronous event. Therefore the notification for success or failure is accomplished using an event callback through the **LINE_GENERATE** line device event.

Automatic Handling by TSP++

Success

If the request is completed successfully, TAPI is notified about the digit completion through a **LINEGENERATETERM_DONE** event code.

Failure

If the request is completed with a non-zero return code, TAPI is notified that the digit generation was canceled through a **LINEGENERATETERM_CANCEL** event code.

RTGenerateTone::GetDuration

DWORD GetDuration() const;

Return Value

This is the duration to generate the tone for. This value was adjusted by TSP++ to make sure it is within the minimum/maximum values presented in the **LINEDEVCAPS** structure.

RTGenerateTone::GetIdentifier

DWORD GetIdentifier() const;

Return Value

This is the unique identifier for this generation request assigned by TAPI.

RTGenerateTone::GetTone

LPLINEGENERATETONE GetTone(unsigned int *iIndex***);**

iIndex	The index position to retrieve the tone structure from.

Return Value

This returns the **LINEGENERATETONE** structure contained at the array position requested. If the array index is out of bounds, NULL is returned.

RTGenerateTone::GetToneArray

TGenerateToneArray* GetToneArray();

Return Value

This returns the array pointer which contains all the tone buffers. It is preferable to use the access methods rather than this direct access pointer.

RTGenerateTone::GetToneCount

unsigned int GetToneCount() const;

Return Value

This returns the number of **LINEGENERATETONE** structures which are contained within this request.

RTGenerateTone::GetToneMode

DWORD GetToneMode() const;

Return Value

This returns the tone mode to generate (**LINETONEMODE_xxx**) requested by the application. It was validated against the **LINEDEVCAPS.dwGenerateToneModes** field.

RTGetPhoneData

This is generated in response to the **TSPI_phoneGetData** function.
This is a special request to TSP++ because this request is specifically waited for on the TAPISRV worker thread that is requesting it. A **WaitForRequest** is performed after the request is queued by the thread. This request *must* be completed eventually or the TAPISRV worker thread (one of many) will never get released.

Automatic Handling by TSP++

Success

None

Failure

None

RTPGetPhoneData::GetBuffer

LPVOID GetBuffer();

Remarks

This buffer points to a block of memory in TAPISRV's address space so it is important not to exceed the size as reported by **GetSize**.

Return Value

Returns a pointer, which the service provider should use to store the phone buffer.

RTPGetPhoneData::GetUploadIndex

DWORD GetUploadIndex() const;

Return Value

The specific phone buffer to retrieve the data from. This was validated by TSP++ against the **PHONECAPS.dwNumGetData** field.

RTPGetPhoneData::GetSize

DWORD GetSize() const;

Return Value

Returns the size of the buffer to store the phone buffer information into.

RTHold

This is generated in response to the **TSPI_lineHold** function.

Automatic Handling by TSP++

Success

None

Failure

None

This function has no additional methods besides the normal **CTSPIRequest** inherited ones.

RTMakeCall

This is generated in response to the **TSPI_lineMakeCall** function.

Special Note

TAPI has a specific requirement related to new calls and the **lineMakeCall** request. It will not forward on any changes to the call information (**LINECALLINFO** or **LINECALLSTATUS**) structures until the asynchronous request associated with the **lineMakeCall** request completes successfully.
One of the common problems with TSPs when handling the **lineMakeCall** request is they send events concerning call states *before* completing the request and the application never sees the events.
To correct this behavior, TSP++ automatically completes the **RTMakeCall** request with a zero return code *if* the attached call attempts to change it's call state before the request was completed.
This means that once the call changes it's call state, you cannot fail the **lineMakeCall** request since it was already reported as successful to TAPI.

Automatic Handling by TSP++

Success

None

Failure

If the request is completed with a non-zero value, then the call object created on behalf of this request is deallocated if it has not yet reported a call state to TAPI (i.e. the application has not seen the call).

RTMakeCall::DialArray

```
TDialStringArray* DialArray();
```

Return Value

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTMakeCall::GetCallParameters

```
LPLINECALLPARAMS GetCallParameters();
```

Return Value

Returns a pointer to the call parameters to use with this newly created call.

RTMakeCall::GetCount

int GetCount() const;

Return Value

Returns the count of dialable addresses passed with this request.

RTMakeCall::GetCountryCode

DWORD GetCountryCode() const;

Return Value

Returns the country code specified for this request. This will normally be zero.

RTMakeCall::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

Return Value

This retrieves a specific **DIALINFO** record that was passed to the request from TAPI.

RTPark

This is generated in response to the **TSPI_linePark** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTPark::GetDialableNumber

LPCTSTR GetDialableNumber() const;

Return Value

This is the dialable address to park the call to if the park mode calls for a directed park. It will be NULL if the request is not for a directed park.

RTPark::GetParkMode

DWORD GetParkMode() const;

Return Value

Returns the requested park mode (**LINEPARKMODE_xxx**).

RTPark::ParkedAddress

LPVARSTRING ParkedAddress();

Return Value

This returns a pointer to a **VARSTRING** that may be modified by the service provider if a non-directed park was requested. The preferable method is to use the **SetParkedAddress** method to modify this structure.

RTPark::SetParkedAddress

bool SetParkedAddress(LPCTSTR pszAddress);

Return Value

This allows the service provider to modify the returning **VARSTRING**. The service provider should fill in the string with the final dialable address of the call as determined by the PBX.

RTPickup

This is generated in response to the **TSPI_linePickup** function.

Automatic Handling by TSP++

Success

None

Failure

If the request is completed with a non-zero return code, then the call appearance created for the pickup is deallocated if it has not been reported to TAPI yet (it has not reported a call-state change).

RTPickup::DialArray

TDialStringArray* DialArray();

Return Value

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTPickup::GetCount

int GetCount() const;

Return Value
Returns the count of dialable addresses passed with this request.

RTPickup::GetCountryCode

DWORD GetCountryCode() const;

Return Value
Returns the country code specified for this request. This will normally be zero.

RTPickup::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

Return Value
This retrieves a specific **DIALINFO** record which was passed to the request from TAPI.

RTPickup::GetGroupID

LPCTSTR GetGroupID() const;

Return Value
This returns the group pickup identifier that was passed to the service provider. It may be NULL.

RTPrepareAddToConference

This is generated in response to the **TSPI_linePrepareAddToConference** function.

Automatic Handling by TSP++

Success

None

Failure

If the request is completed with a non-zero return code, then the created consultation call appearance is deallocated if it has not been reported to TAPI yet (it has not reported a call-state change).

RTPrepareAddToConference::GetCallParameters

LPLINECALLPARAMS GetCallParameters();

Return Value

This returns the **LINECALLPARAMS** to adjust the created consultation call with pre-set features. The derived service provider must handle this operation.

RTPrepareAddToConference::GetConferenceCall

CTSPIConferenceCall* GetConferenceCall() const;

Return Value

Returns the conference call object that this request relates to.

RTPrepareAddToConference::GetConsultationCall

CTSPICallAppearance* GetConsultationCall() const;

Return Value

Returns the consultation call that is being added to the conference.

RTRedirect

This is generated in response to the **TSPI_lineRedirect** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTRedirect::DialArray

TDialStringArray* DialArray();

Return Value

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTRedirect::GetCount

int GetCount() const;

Return Value
Returns the count of dialable addresses passed with this request.

RTRedirect::GetCountryCode

DWORD GetCountryCode() const;

Return Value
Returns the country code specified for this request. This will normally be zero.

RTRedirect::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

Return Value
This retrieves a specific **DIALINFO** record that was passed to the request from TAPI.

RTReleaseUserInfo

This is generated in response to the **TSPI_lineReleaseUserInfo** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ frees the top buffer in the call's user-user information array. If there is additional information in the array, it is posted to TAPI through the **LINECALLINFO** structure. (see the **CTSPICallAppearance::ReleaseUserUserInfo** for more information).

Failure

None
This function has no additional methods besides the normal **CTSPIRequest** inherited ones.

RTRemoveFromConference

This is generated in response to the **TSPI_lineRemoveFromConference** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ removes the call from the conference and forces a breakdown of the conference if only one call remains. The call state of the removed call is *not* changed.

Failure

None

RTRemoveFromConference::GetCallToRemove

CTSPICallAppearance* GetCallToRemove() const;

Return Value

This returns the call object to remove from the conference.

RTRemoveFromConference::GetConferenceCall

CTSPIConferenceCall* GetConferenceCall() const;

Return Value

This returns the conference call object that this request is attached to.

RTSecureCall

This is generated in response to the **TSPI_lineSecureCall** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the call's flags (**LINECALLSTATUS.dwCallParamFlags** to indicate that the call is *secure* (the **LINECALLPARAMFLAGS_SECURE** bit). TAPI is notified about the change. Note that this has no effect on the call, TAPI or TSP++. It is assumed that the derived provider sent the appropriate commands to ensure that the call cannot be accidentally dropped or interrupted.

Failure

None
This function has no additional methods besides the normal **CTSPIRequest** inherited ones.

RTSendUserInfo

This is generated in response to the **TSPI_lineSendUserUserInfo** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTSendUserUserInfo::GetSize

DWORD GetSize() const;

Return Value

Returns the size of the user-user information block.

RTSendUserUserInfo::GetUserUserInfo

LPCSTR GetUserUserInfo() const;

Return Value

Returns the user-user information buffer to send to the other side of the call. The size was validated against the **LINEDEVCAPS.dwUUISendUserUserInfoSize** field.

RTSetAgentActivity

This is generated in response to the **lineSetAgentActivity** function using the **JTAPROXY** handler.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the current agent's activity to be the specified value.

Failure

None

RTSetAgentActivity:: GetActivity

```
DWORD GetActivity() const;
```

Return Value

Returns the requested activity code. This was *not* validated by TSP++ so if the provider cannot accept non-supported activity codes it should check the value using the **CTSPIDevice::DoesAgentActivityExist** method.

RTSetAgentGroup

This is generated in response to the **lineSetAgentGroup** function using the **JTAPROXY** handler.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the current agent's group settings to be the specified groups. If the group array is empty, the agent is automatically logged off.

Failure

None

RTSetAgentGroup::GetCount

unsigned int GetCount() const;

Return Value

This returns the total number of groups contained within the group array. The value can be zero if a logoff request is being initiated by an application.

RTSetAgentGroup::GetGroup

const TAgentGroup* GetGroup(unsigned int *iIndex***) const;**

iIndex The array index of the group structure to retrieve.

Return Value

Pointer to the group structure requested. If the array index is invalid, NULL is returned.

RTSetAgentGroup::GetGroupArray

TAgentGroupArray* GetGroupArray();

Return Value

This returns a pointer to the array holding all the requested agent groups. The groups may be modified, removed, etc. It is preferable to use the access methods defined to access the information rather than this direct access approach.

RTSetAgentState

This is generated in response to the **lineSetAgentState** function using the **JTAPROXY** handler.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the current agent's state to be the specified value.

Failure

None

RTSetAgentState::GetAgentState

DWORD GetAgentState() const;

Return Value

The requested agent state (**LINEAGENTSTATE_xxx**) for this address.

RTSetAgentState::GetNextAgentState

DWORD GetNextAgentState() const;

Return Value

The requested *next* agent state for this address. This typically is the state that the agent will be in once the current call is released.

RTSetAgentState::SetAgentState

void SetAgentState(DWORD dwState);

dwState **LINEAGENTSTATE** to set into the request

Remarks

This allows the state value to be adjusted by the service provider before completing the request. The value is not validated by TSP++.

RTSetAgentState::SetNextAgentState

void SetNextAgentState(DWORD dwState);

dwState **LINEAGENTSTATE** to set into the request

Remarks

This allows the *next* state value to be adjusted by the service provider before completing the request. The value is not validated by TSP++.

RTSetButtonInfo

This is generated in response to the **TSPI_lineSetButtonInfo** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the specified button's function, mode, and text to the requested values. TAPI is notified of the change.

Failure

None

RTSetButtonInfo::GetButtonFunction

DWORD GetButtonFunction() const;

Return Value

This returns the requested button function for the button (**PHONEBUTTONFUNCTION_xxx**).

RTSetButtonInfo::GetButtonInfo

LPPHONEBUTTONINFO GetButtonInfo();

Return Value

This returns a pointer to the actual **PHONEBUTTONINFO** structure passed with the request. It is preferable to use the defined access methods instead of this direct access method.

RTSetButtonInfo::GetButtonLampID

DWORD GetButtonLampID() const;

Return Value

This returns the unique button/lamp identifier which identifies the specific button this request is referring to.

RTSetButtonInfo::GetButtonMode

DWORD GetButtonMode() const;

Return Value

This returns the requested button mode for the button (**PHONEBUTTONMODE_xxx**).

RTSetButtonInfo::GetButtonText

LPCTSTR GetButtonText() const;

Return Value

This returns the button text to be set into the button.

RTSetButtonInfo::GetDevSpecificInfo

LPVOID GetDevSpecificInfo();

Return Value

This returns the device-specific information that was passed with the request. No interpretation is done to this information and it may be NULL or invalid.

RTSetCallData

This is generated in response to the **TSPI_lineSetCallData** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the call data for the specified call to be the requested value.

Failure

None

RTSetCallData::GetData

LPCVOID GetData() const;

Return Value

This returns the pointer to the data buffer that represents the opaque call data to set into the call. The service provider should not attempt to interpret this data since it is application-specific. This block may be NULL to clear the call data field.
The buffer size was validated against the **LINEADDRESSCAPS.dwMaxCallDataSize** field.

RTSetCallData::GetSize

DWORD GetSize() const;

Return Value

The size of the passed block. This may be zero.

RTSetCallParams

This is generated in response to the **TSPI_lineSetCallParams** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the bearer mode and the dialing information to the requested values.

Failure

None

RTSetCallParams::GetBearerMode

DWORD GetBearerMode() const;

Return Value

This returns the requested bearer mode for the call.

RTSetCallParams::GetDialParams

LPLINEDIALPARAMS GetDialParams();

Return Value

This returns the pointer to the **LINEDIALPARAMS** structure passed with the request.

RTSetCallParams::GetMinDataRate

DWORD GetMinDataRate() const;

Return Value

This returns the minimum requested data rate for the call

RTSetCallParams::GetMaxDataRate

DWORD GetMaxDataRate() const;

Return Value

This returns the maximum requested data rate for the call

RTSetCallParams::SetBearerMode

void SetBearerMode(DWORD dwBearerMode);

Return Value

This allows the service provider to adjust the bearer mode before completing the request.

RTSetCallParams::SetDataRate

void SetDataRate(DWORD dwMinRate, DWORD dwMaxRate);

Return Value

This allows the service provider to adjust the min/max data rates before completing the request.

RTSetCallTreatment

This is generated in response to the **TSPI_lineSetCallTreatment** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the call treatment value into the **LINECALLINFO** structure and notifies TAPI.

Failure

None

RTSetCallTreatment::GetCallTreatment

```
DWORD GetCallTreatment() const;
```

Return Value

This returns the requested call treatment for the call (**LINECALLTREATMENT_xxxx**).

RTSetDisplay

This is generated in response to the **TSPI_phoneSetDisplay** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the phone's display contents and notifies TAPI.

Failure

None

RTSetDisplay::GetBufferPtr

LPCTSTR GetBufferPtr() const;

Return Value

This returns the buffer which is to replace the display at the specified col/row position.

RTSetDisplay::GetBufferSize

DWORD GetBufferSize() const;

Return Value

This is the size of the data pointed at by **GetBufferPtr**.

RTSetDisplay::GetColumn

unsigned int GetColumn() const;

Return Value

This returns the starting column where the display is to be changed.

RTSetDisplay::GetRow

unsigned int GetRow() const;

Return Value

This returns the starting row to change on the phone's display.

RTSetGain

This is generated in response to the **TSPI_phoneSetGain** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the phone's gain to the specified value.
None

Failure

None

RTSetGain::GetGain

DWORD GetGain() const;

Return Value

This returns the request gain value.

RTSetGain::GetHookswitchDevice

DWORD GetHookswitchDevice() const;

Return Value

This returns the hookswitch to adjust. It will be a constant from the **PHONEHOOKSWITCHDEV_xxx** bits. The value was validated against the **PHONECAPS.dwGainFlags** field.

RTSetHookswitch

This is generated in response to the **TSPI_phoneSetHookswitch** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the phone's hookswitch to the specified value.

Failure

None

RTSetHookSwitch::GetHookswitchDevice

DWORD GetHookswitchDevice() const;

Return Value

This returns the hookswitch to adjust. It will be a constant from the **PHONEHOOKSWITCHDEV_xxx** bits. The value was validated against the **PHONECAPS.dwSettableXXX** fields if they are defined.

RTSetHookSwitch::GetHookswitchState

DWORD GetHookswitchState() const;

Return Value

This returns the new value of the hookswitch (**PHONEHOOKSWITCHMODE_**). The value was validated against the **PHONECAPS.dwSettableXXX** fields if they are defined.

RTSetLampInfo

This is generated in response to the **TSPI_phoneSetLamp** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the phone's lamp state to the specified value.

Failure

None

RTSetLampInfo::GetButtonLampID

DWORD GetButtonLampID() const;

Return Value

This returns the unique button/lamp identifier that identifies the specific lamp this request is referring to.

RTSetLampInfo::GetLampMode

DWORD GetLampMode() const;

Return Value

This returns the requested lamp mode for the lamp (**PHONELAMPMODE_xxx**).

RTSetLineDevStatus

This is generated in response to the **TSPI_lineSetDevStatus** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ alters the line's device status flags in the requested fashion. TAPI is notified.

Failure

None

RTSetLineDevStatus::GetStatusBitsToChange

```
DWORD GetStatusBitsToChange() const;
```

Return Value

This returns the OR'd list of bits to change (**LINEDEVSTATUSFLAGS_xxx**). This list was validated against the **LINEDEVCAPS. dwSettableDevStatus** field.

RTSetLineDevStatus::TurnOnBits

```
bool TurnOnBits() const;
```

Return Value

TRUE if the bits are to be turned on, FALSE if the bits are to be turned off.

RTSetMediaControl

This is generated in response to the **TSPI_lineSetMediaControl** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the media control list for all affected objects to match the requested values.

Failure

None

RTSetMediaControl::GetAddress

CTSPIAddressInfo* GetAddress() const;

Return Value

This returns the target address of the command. This may be NULL if all addresses on a line are to be targeted. Note that the **GetCall** method should be called to determine if only a single call is to be targeted.

RTSetMediaControl::GetCall

CTSPICallAppearance* GetCall() const;

Return Value

This returns the specific call to target. This may be NULL If all calls on an address are to be changed.

RTSetMediaControl::GetLine

CTSPILineConnection* GetLine() const;

Return Value

This returns the line to alter. Note that the **GetAddress** and **GetCall** methods should be called to see if this change is for a specific address or line.

RTSetMediaControl::GetMediaControlInfo

TSPIMEDIACONTROL* GetMediaControlInfo();

Return Value

This returns the new media control information for the target.

RTSetPhoneData

This is generated in response to the **TSPI_phoneSetData** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTSetPhoneData::GetBuffer

LPCVOID GetBuffer();

Return Value

This is the data to store into the phone's buffer. It can be NULL to remove all information from the buffer.

RTSetPhoneData::GetDownloadIndex

DWORD GetDownloadIndex() const;

Return Value

The specific phone buffer to retrieve the data from. This was validated by TSP++ against the **PHONECAPS.dwNumSetData** field.

RTSetPhoneData::GetSize

DWORD GetSize() const;

Return Value

Returns the size of the buffer to store into the phone.

RTSetQualityOfService

This is generated in response to the **TSPI_lineSetCallQualityOfService** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the new **FLOWSPEC** values into the call object.

Failure

None

RTSetQualityOfService:: GetReceivingFlowSpec

LPCVOID GetReceivingFlowSpec() const;

Return Value

Pointer to memory block containing a WinSock2 **FLOWSPEC** structure followed by optional service provider-specific data.

RTSetQualityOfService:: GetReceivingFlowSpecSize

DWORD GetReceivingFlowSpecSize() const;

Return Value

Size of the data block

RTSetQualityOfService:: GetSendingFlowSpec

LPCVOID GetSendingFlowSpec() const;

Return Value

Pointer to memory block containing a WinSock2 **FLOWSPEC** structure followed by optional service provider-specific data.

RTSetQualityOfService:: GetSendingFlowSpecSize

DWORD GetSendingFlowSpecSize() const;

Return Value

Size of the data block.

RTSetRing

This is generated in response to the **TSPI_phoneSetRing** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the new ringer style and volume into the phone object.

Failure

None

RTSetRing::GetRingMode

DWORD GetRingMode() const;

Return Value

This returns the desired ring mode for the phone unit. It is guaranteed to be a value from zero to the number of ring modes in **PHONECAPS.dwRingModes**.

RTSetRing::GetVolume

DWORD GetVolume() const;

Return Value

This returns the desired volume level for the ringer.

RTSetTerminal

This is generated in response to the **TSPI_lineSetTerminal** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ sets the new terminal modes into the internal data structures for each affected device and reports that they have been altered to TAPI.

Failure

None

RTSetTerminal::Enable

bool Enable() const;

Return Value

If TRUE, the specified event classes are to be routed to or from the specified terminal. If FALSE, these events are not routed to or from the specified terminal.

RTSetTerminal::GetAddress

CTSPIAddressInfo* GetAddress() const;

Return Value

This returns the target address of the command. This may be NULL if all addresses on a line are to be targeted. Note that the **GetCall** method should be called to determine if only a single call is to be targeted.

RTSetTerminal::GetCall

CTSPICallAppearance* GetCall() const;

Return Value

This returns the specific call to target. This may be NULL If all calls on an address are to be changed.

RTSetTerminal::GetLine

CTSPILineConnection* GetLine() const;

Return Value

This returns the line to alter. Note that the **GetAddress** and **GetCall** methods should be called to see if this change is for a specific address or line.

RTSetTerminal::GetTerminalID

DWORD GetTerminalID() const;

Return Value

The terminal device identifier where the given events are to be or not to be routed. This was validated using the **LINEDEVCAPS.dwNumTerminals** field.

RTSetTerminal::GetTerminalModes

DWORD GetTerminalModes() const;

Return Value

The class(es) of low level events to be routed to the given terminal (**LINETERMMODE_xxx**).

RTSetupConference

This is generated in response to the **TSPI_lineSetupConference** function.

Automatic Handling by TSP++

Success

None

Failure

If the request is completed with a non-zero return code, the created consultation call is deallocated if it has not yet been shown to TAPI (it has not sent a callstate event).

RTSetupConference::GetCallParameters

LPLINECALLPARAMS GetCallParameters();

Return Value

This returns the **LINECALLPARAMS** to adjust the created consultation call with pre-set features. The derived service provider must handle this operation.

RTSetupConference::GetConferenceCall

CTSPIConferenceCall* GetConferenceCall() const;

Return Value

Returns the conference call object that this request relates to.

RTSetupConference::GetConsultationCall

CTSPICallAppearance* GetConsultationCall() const;

Return Value

Returns the consultation call that is being added to the conference.

RTSetupConference::GetInitialPartyCount

DWORD GetInitialPartyCount() const;

Return Value

This returns the applications estimate on how many parties will be in the conference. This is not used by TSP++ but is delivered in case the service provider allocates additional memory or needs to do some special work with the hardware.

RTSetupConference::GetOriginalCall

CTSPICallAppearance* GetOriginalCall() const;

Return Value

This returns the original call that is starting the conference. This is normally a two-party call in the **LINECALLSTATE_CONNECTED** state.

RTSetupTransfer

This is generated in response to the **TSPI_lineSetupTransfer** function.

Automatic Handling by TSP++

Success

None

Failure

If the request is completed with a non-zero return code, the created consultation call is deallocated if it has not yet been shown to TAPI (it has not sent a callstate event).

RTSetupTransfer::GetCallParameters

```
LPLINECALLPARAMS GetCallParameters();
```

Return Value

This returns the **LINECALLPARAMS** to adjust the created consultation call with pre-set features. The derived service provider must handle this operation.

RTSetupTransfer::GetCallToTransfer

```
CTSPICallAppearance* GetCallToTransfer() const;
```

Return Value

This returns the call that is to be transferred.

RTSetupTransfer::GetConsultationCall

```
CTSPICallAppearance* GetConsultationCall() const;
```

Return Value

This returns the newly created consultation call that may be dialed to reach a target for the transfer event.

RTSetVolume

This is generated in response to the **TSPI_phoneSetHookswitchVolume** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, TSP++ changes the phone's volume level to the specified value.
None

Failure

None

RTSetVolume::GetHookswitchDevice

DWORD GetHookswitchDevice() const;

Return Value

This returns the hookswitch to adjust. It will be a constant from the **PHONEHOOKSWITCHDEV_xxx** bits. The value was validated against the **PHONECAPS.dwVolumeFlags** field.

RTSetVolume::GetVolume

DWORD GetVolume() const;

Return Value

This returns the request volume value.

RTSwapHold

This is generated in response to the **TSPI_lineSwapHold** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, the *call type* is swapped between the two calls in case this is a swap between a consultation call and the original call. In effect, the consultation call becomes a real call (now onHold) and the original call becomes the consultation call.

Failure

None

RTSwapHold::GetActiveCall

CTSPICallAppearance* GetActiveCall() const;

Return Value

This returns the current active call to swap the holding call with.

RTSwapHold::GetHoldingCall

CTSPICallAppearance* GetHoldingCall() const;

Return Value

This returns the current call which is on hold.

RTUncompleteCall

This is generated in response to the **TSPI_lineUncompleteCall** function.

Automatic Handling by TSP++

Success

If the request is completed successfully, the call completion record stored in the line owner is removed and marked invalid.

Failure

None

RTUncompleteCall::GetRTCompleteCall

```
RTCompleteCall* GetRTCompleteCall() const;
```

Return Value

Returns the original **RTCompleteCall** request which is waiting for the call completion event.

RTUnhold

This is generated in response to the **TSPI_lineUnhold** function.

Automatic Handling by TSP++

Success

None

Failure

None

This function has no additional methods besides the normal **CTSPIRequest** inherited ones.

RTUnpark

This is generated in response to the **TSPI_lineUnpark** function.

Automatic Handling by TSP++

Success

None

Failure

None

RTUnpark::DialArray

TDialStringArray* DialArray();

Return Value

This returns a pointer to the dial array with all the **DIALINFO** structures present. It is preferable to use the **GetCount** and **GetDialableNumber** to access this information.

RTUnpark::GetCount

int GetCount() const;

Return Value

Returns the count of dialable addresses passed with this request.

RTUnpark::GetCountryCode

DWORD GetCountryCode() const;

Return Value

Returns the country code specified for this request. This will normally be zero.

RTUnpark::GetDialableNumber

DIALINFO* GetDialableNumber(unsigned int *iIndex***) const;**

iIndex Index from (0-**GetCount**) to retrieve **DIALINFO** from.

Return Value

This retrieves a specific **DIALINFO** record that was passed to the request from TAPI.

RTUnpark::GetGroupID

LPCTSTR GetGroupID() const;

Return Value

This returns the group pickup identifier that was passed to the service provider. It may be NULL.

Thread Pool Manager template

Threads that process events from the switch have to be managed by each developed service provider. A number of event thread models can be employed in a server TSP's design. The particular model used depends on the nature of the PBX or ACD involved, and the scaling requirements of the TSP itself. Any of the following models can be used with TSP++ V3.0 depending on what the design requirements are.

Single thread; single event at a time

This is the simplest of all models and is the model used in TSP++ V2.x. In this model, the TSP has a single loop in which it accepts a single incoming event from the switch, which it immediately services. Because the server cannot accept a new event until the previous event is completed, the events are always synchronized properly with regard to the switch. This model is inappropriate for a large scale TSP because it cannot take advantage of multiprocessor systems (non-scalable) and it could quickly be overrun by server events if the PBX has a large number of devices connected to it.

One thread per event

Probably the most commonly used model for server applications, this model is also the fastest one for models that intend to service fewer than 16 events at a time. In this model, the TSP executes a single loop that accepts incoming events and then creates a worker thread for each incoming event. This model is relatively easy to implement, and is used in the ATSP32 sample given out by Microsoft. A major problem with this model with regards to PBX modeling is that many events must be synchronized – in order words, some events must be completely finished before the next related event can be processed. If this is not done, the TSP can quickly become out of synch with the switch depending on how quickly threads are scheduled by Windows NT. Another drawback to this model is that it does not scale well to serving large numbers of clients. This is because the larger numbers of threads impose greater demands on system services, and the context switching required for the operating system to create and execute each thread can impose a significant penalty on CPU usage.

Thread Pool

In this model, a series of threads are created for the specific purpose of managing incoming events. When an event is completed, the thread cycles around to manage the next incoming event. This method still suffers from the synchronization issues listed above but solves the problem of too many simultaneous threads. Since the total count of threads is limited to some figure (based on system resources available) and we only incur the creation time penalty once, this model is much better suited for a larger number of incoming requests.

Thread Pool with synchronization

This model is identical to the previous, with the exception that each request is tagged with a synchronization key that is checked before allowing the request to be processed. This keeps requests in their intended order and insures that a request is not processed

before the previous related request has been completely finished. This is the model used by the TSP++ thread pool manager template.

Thread Pool Manager Template class

To use this class, you must *#include <poolmgr.h>* in your header file which uses the class. This is not done automatically.

This template class generates a C++ object which supports a *key* entry which is used to synchronize requests with each other – as long as the key is *locked*, any events which share that key will not be processed by worker threads. Once the key is unlocked, then next pending event for that key (if any exists in the queue) will be processed. If another event is queued which is associated with a different key element that is not locked, it will be processed immediately – while the other request is running. This *key* is a parameter class to the template.
The data element passed to and from the thread pool is a parameter class to the template so it may be any object, structure, or data entry desired. It represents the event received from the switch.
When an event is queued to the thread pool, a pointer to the line or phone device that will own this event is also given. This pointer is then used to dispatch the event using one of the created worker threads when the event can be processed.

Template Definition

```
template <class _KEY, class _REQ> CThreadPoolMgr(int iMaxCount=0);
```

To use the template, simply declare a variable in the device object (**CTSPIDevice**) which is derived from the template and is given the key and event object types. For an example on this, see the section on *Thread Pool Management* in the user's guide.

TPM_DelEvent

The TPM_DelEvent template function is used to destroy the event object once a thread has dispatched it through the service provider code. Since the event parameter could be an object, structure, or native C/C++ type the requirements for destroying it might vary. To make the pool manager as flexible as possible, this template function is used to destroy the elements. By default, it will generate a function which will call the C++ **delete** function on the event. If this is not acceptable, then the derived service provider should supply a specific **TPM_DelEvent** inline function which takes the event object as a parameter and perform its own destruction method.
For example, if the event object were a DWORD, then no destruction would be necessary since it was not allocated. In this case, **TPM_DelEvent** would be overridden to do nothing with a DWORD type:

```
inline void TPM_DelEvent(DWORD dwEvent)
```

```
{
        // Do nothing since this is a dword that is not allocated.
}// TPM_DelEvent
```

This implementation will cause the template function to *not* expand and therefore this inline function will be used instead.
Note: it is very important that the replacement function is defined *before* the template class is expanded (before the data declaration in the **CTSPIDevice** overridden object). Also, the function should be inline since it will generally only be called in one spot.

TPM_CanRunEvent

This template function is used to change the way that the key is used to determine whether an event can run. In many cases, two events for the same target can run simultaneously since they don't conflict with each other. An example might be an event related to a lamp change and one related to a display change. They are independent of each other and can be run together.
If the key is simply used as a map, these two events would be synchronized since they refer to the same basic object. This can be changed using the template function **TPM_CanRunEvent**. The prototype is:

```
inline bool TPM_CanRunEvent(_REQ req, _KEY key, bool fIsLocked);
```

This function is called when the thread pool has an available worker thread and is trying to find work for it to do. It calls this function and passes the event which it is currently examining, the key for the event, and whether the lock key is currently set due to another thread working with the key. If a non-zero response is returned from this function, then the current thread will be released to run the event through the service provider. If a zero result is returned, the event will be kept in-queue.
The default implementation returns a non-zero result only if the key is *not* locked. In other words, it forces all events for the same lock key to be synchronized. This is the safest implementation.
If there are events which can run together, this function template can be overridden using an inline function in the derived provider which takes the appropriate parameters and can decide how to run the events.

Constructor

CthreadPoolMgr	Constructor for the pool manager

Operations - Public Members

Add	Adds a new request to the event queue.

CThreadPoolMgr::Add

bool Add(CTSPIConnection* *pConn*, **_KEY** *key*, **_REQ** *event*);

pConn	Line or phone object which will service the event
key	Synchronization key
event	Event received

Remarks

This method adds the given event to the event queue and eventually causes a worker thread to service the event.

Return Value

TRUE if the event was added to the queue. FALSE if an exception occurred.

CThreadPoolMgr::CThreadPoolMgr

CThreadPoolMgr(int *iMaxCount*=**0);**

iMaxCount	Maximum number of worker threads to allocate

Remarks

This is the constructor for the pool manager. If zero is passed for the *iMaxCount* parameter, then the pool template will allocate up to 1 thread per every 32M of RAM on the server with a maximum of 32 threads per processor (whichever is smaller).

This has been determined in testing to be an effective gauge of how well the server will perform given memory and processor counts.

Note that worker threads are created as needed and *not* initially when the pool is initialized.

TStream binary stream class

TSP++ provides for generic data storage to the Windows Registry using the **ReadProfileString**, **ReadProfileDWord**, **WriteProfileString**, and **WriteProfileDWord** methods. This may be used for simple configuration providers that don't need to store large amounts of configuration for their implementation.
To help support providers which needs to store off information of the hardware, TSP++ also implements a binary stream class. This class is used by the SPLUI and SPLIB libraries to pass data back and forth in a compatible fashion using simple insertion (<<) and extraction (>>) operators.
The **TStream** class implemens an iostream-type class which supports binary information storage (unlike iostreams which is text-based). It supports Unicode strings and all basic C data types. The base class (**TStream**) described here is a *virtual base class*. This means that it cannot be used as implemented – it must be derived from to be usefull. TSP++ implements a registry stream for storing persistent information into the Windows Registry in the *regstream.h* file. This is the default stream used for storing configuration. This section is included in case the developer wants to store the configuration in a different area other than the Registry. For a sample on overriding the **TStream** class, see the *User's Guide* section on *Persistent Object Information*.

To override this class, you must *#include <spbstrm.h>* in your header file which uses the class. This is not done automatically.

Implementation Notes:
All strings are stored as 16-bit Unicode strings – even if the Ansi library is used.
No attempt is made in the registry stream class to byte-order the information as it is assumed the same machine will be reading and writing the data. TAPI doesn't support cross-network installation of TAPI service providers.
The current usage doesn't mix read and write functions -

Overridables - Public Members

close	Called to close the stream when all data has been read or written.
open	Called to open the stream and prepare it for being read or written to.
read	Called to read a series of bytes from the stream.
write	Called to write information into the stream.

Operations - Public Members

operator<<(bool)	Insertion operator for the **bool** data type.
operator<<(CString)	Insertion operator for the MFC **CString** data type.
operator<<(double)	Insertion operator for the **double** data type.
operator<<(float)	Insertion operator for the **float** data type.
operator<<(int)	Insertion operator for the **int** data type.

operator<<(long)	Insertion operator for the **long** data type.
operator<<(short)	Insertion operator for the **short** data type.
operator<<(std::string)	Insertion operator for the STL **string** data type.
operator<<(std::wstring)	Insertion operator for the STL **wstring** data type.
operator<<(unsigned int)	Insertion operator for the **unsigned int** data type.
operator<<(unsigned long)	Insertion operator for the **unsigned long** data type.
operator<<(unsigned short)	Insertion operator for the **unsigned short** data type.
operator>>(bool)	Extraction operator for the **bool** data type.
operator>>(CString)	Extraction operator for the MFC **CString** data type.
operator>>(double)	Extraction operator for the **double** data type.
operator>>(float)	Extraction operator for the **float** data type.
operator>>(int)	Extraction operator for the **int** data type.
operator>>(long)	Extraction operator for the **long** data type.
operator>>(short)	Extraction operator for the **short** data type.
operator>>(std::string)	Extraction operator for the STL **string** data type.
operator>>(std::wstring)	Extraction operator for the STL **wstring** data type.
operator>>(unsigned int)	Extraction operator for the **unsigned int** data type.
operator>>(unsigned long)	Extraction operator for the **unsigned long** data type.
operator>>(unsigned short)	Extraction operator for the **unsigned short** data type.

TStream::close

virtual void close();

Remarks

This method is called when the stream operations are completed. This should flush any written information into the stream.

TStream::open

virtual bool open();

Remarks

This method is called when the library is preparing to load information or write information concerning object persistence. Once the stream is opened, the library may read or write to the stream immediately.

Return Value

TRUE if the stream is ready for read/write operations; FALSE if there is an error. If FALSE is returned, the library will throw an exception.

TStream::read

virtual bool read(void* *pBuff*, **unsigned int** *size*) **= 0;**

pBuff	Buffer to fill within information.
size	Number of bytes to read from the stream.

Remarks

This method is called to retrieve information from the stream. It is used by all of the extraction operators.

Return Value

TRUE if the information was retrieved; FALSE if there is an error. If FALSE is returned, the library will throw an exception.

TStream::write

virtual bool write(void* *pBuff*, **unsigned int** *size*) **= 0;**

pBuff	Buffer to store into the stream.
size	Number of bytes to write into the stream.

Remarks

This method is called to store information into the stream. It is used by all of the insertion operators.

Return Value

TRUE if the information was stored; FALSE if there is an error. If FALSE is returned, the library will throw an exception.

Data Structures

The following are some of the common data structure definitions used for miscellaneous internal TSP++ operations.

Most of these structures are internal to the operation of TSP++ and do not need to be understood or managed by the derived service provider code. They are documented here for clarity of the source code and debugging purposes only.

CALLIDENTIFIER

This structure is used to hold network information for a call. This is sometimes referred to as *caller-id* but in TAPI, is also used for various parties that interacted with the call (i.e. called party, calling party, redirecting party, etc.)

Data Member	Description
strPartyID	Numeric digits formatted (if possible) in canonical format.
strPartyName	Textual name of the person at the given number.

DEVICECLASSINFO

This structure is used to maintain information about *Device Class* properties that are associated with each object type. These device class structures are returned during the processing for **LINEDEVCAPS** and **TSPI_lineGetID**.

Data Member	Description
strName	Name of the device (“tapi/line”)
dwStringFormat	String format (STRINGFORMAT_xxx)
lpvData	Data associated with device class (may be NULL)
dwSize	Size of data
hHandle	Win32 object handle that is copied to requesting process space.

DIALINFO

This structure is used to manage dialing information from TAPI.

Data Member	Description
fIsPartialAddress	The address was “partial”, and the call should not leave the dialing state. This was indicated in the dialable address by the trailing “;”. The “;” is removed from the dial string.
strNumber	The number removed from the dialable address. This is a single address, with all invalid characters (see above list for valid characters) removed.
strName	The name which was attached to the dialable address

strSubAddress	The ISDN sub-address which was attached to the dialable address.

Rather than forcing the derived service provider class to break the information out of the dialable address format, the **CServiceProvider::CheckDialableAddress** method is provided to do all the work. The method is used to break a dialable address into its component parts. The method then returns an object array with potentially multiple **DIALINFO** structures in it. Each **DIALINFO** represents a single address found in a dialable address string. This array is then passed to the requests which need addresses (**lineMakeCall**, **lineRedirect**, **linePark**, etc.)

EXTENSIONID

This is a combination of a **LINEEXTENSIONID** and **PHONEEXTENSIONID** structure so the **CTSPIConnection** object can manage both structures within one. For more information on this structure and its usage, see the TAPI documentation on the functions **TSPI_lineGetExtensionID** and **TSPI_phoneGetExtensionID**.

Data Member	Description
dwExtensionID0	First 32-bit portion of the 128-bit GUID describing the extensions.
dwExtensionID1	Second 32-bit portion of the 128-bit GUID.
dwExtensionID2	Third 32-bit portion of the 128-bit GUID.
dwExtensionID3	Fourth 32-bit portion of the 128-bit GUID.

LOCATIONINFO

This structure stores all the required country information when the current location changes. It gives the provider enough information to convert from dialable to canonical format and is used by the **CServiceProvider::ConvertDialableToCanonical** method for formatting caller-id information.

Data Member	Description
dwCurrentLocation	Current defined location – if no application has not set the location using the **lineSetCurrentLocation** function then this will be the default location for the system.
strCountryCode	Country code for this location, this is the first series of digits used in long distance calls within the country (i.e. **1** in USA)
strAreaCode	Area code for the state or area – this may be blank for some countries as they are too small to be divided into multiple areas.
strIntlCode	International dialing code – this is the series of digits which must be dialed in order to call another country.
strLocalAccess	Digits which are dialed to get a local access line (**9**on many PBX systems).
strLongDistanceAccess	Digits which are dialed to get a long distance access line (**9**on many PBX systems).

strCallWaiting	This is the string of digits dialed to disable call-waiting on the line.

SIZEDDATA

This is an object with an embedded pointer of an undefined type (i.e. void) and 32-bit size value. It tracks the pointer and its contents and automatically deletes the information when the structure is deleted. This is an internal data structure and generally does not need to be dealt with in the derived service provider.
This structure is used to store user-user information, QOS information, call data, and other variably-sized pieces of data.

Member Function	Description
Free	Frees the data pointer and sets it to NULL
GetSize	Returns the size of the data contained within the pointer.
GetPtr	Returns a readable constant pointer to the data
GetWPtr	Returns a pointer that may be modified. The size of the data cannot be changed.
SetPtr	Sets the pointer to a new value and size. Use this to change the contents of the pointer.

TERMINALINFO

This structure defines a terminal to our line connection. Each added terminal will have a structure of this type added to the *arrTerminals* list in the line connection. Each address and call will have a DWORD list describing the mode of the terminal info (superceding the **dwMode** value here).

Data Member	Description
strName	Name of the terminal device
Capabilities	**LINETERMCAPS** structure embedded within the structure.
dwMode	Current mode of the terminal device.

TExtVersionInfo

This pulls together all the extended version information so it can be stored in the line and phone objects as a dynamically allocated element. Since many providers don't support this, it is only allocated when the provider uses the **SetExtVersionInfo** method to create it. This structure should not be directly addressed – instead use the provided methods of the **CTSPIConnection** object to retrieve and set information in this structure.

Data Member	Description
dwMinExtVersion	Minimum negotiable version for extended telephony features
dwSelectedExtVersion	Currently selected extension version.

dwMaxExtVersion	Maximum negotiable version for extended telephony features.
ExtensionID	Embedded **EXTENSIONID** structure (see above).

TIMEREVENT

This structure is used by the **CTSPICallAppearance** object to manage interval timer events that are checked within the **OnInternalTimer** method. This includes things such as digit gathering, monitoring, tone detection, and media events. This structure is internal to TSP++ and is automatically managed and used.

Data Member	Description
iEventType	Type of event which needs to be checked
dwEndTime	**GetTickCount** time when this event expires.
dwData1	Optional data used within event checking code.
dwData2	Optional data used within event checking code.

TSPIDIGITGATHER

The **CTSPICallAppearance** object uses this structure to manage and maintain the digit gathering process.

Data Member	Description
dwEndToEndID	Unique identifier for TAPI
dwDigitModes	**LINEDIGITMODE_xxx** values to watch for
dwSize	Number of digits to collect before it is completed.
dwCount	Number of digits that have been collected.
dwFirstDigitTimeout	mSec timeout to wait for the first digit.
dwInterDigitTimeout	mSec timeout to wait for each digit.
dwLastTime	mSec timer value when last digit arrived.
lpBuffer	Buffer where digits are being collected.
strTerminationDigits	Digits which will terminate the collection.

TSPIFORWARDINFO

Single forwarding information structure. This is placed into the *arrForwardInfo* array for each forward entry found in the forward list.

Data Member	Description
dwTotalSize	Total size of all the forwarding information contained within this object.
dwForwardMode	Forwarding mode for this request (LINEFORWARDMODE_xxx). This was validated against the forwarding types available in the LINEADDRESSCAPS dwForwardModes field.

dwDestCountry	Destination country for the address array
arrCallerAddress	Caller address information for specific forwarding mode. Depending on the forwarding mode, this array might be empty. If it has entries, they will be of type DIALINFO.
arrDestAddress	Destination address information. This should generally always have at least one address (and should generally only have one). The data will be of type DIALINFO.

TSPIMEDIACONTROL

This structure contains the data used for managing media control events within the service provider. This is passed to the **SetMediaControl** method of the various objects when a media event occurs.

Data Member	Description
arrDigits	Array of **LINEMEDIACONTROLDIGIT** structures.
arrMedia	Array of **LINEMEDIACONTROLMEDIA** structures.
arrTones	Array of **LINEMEDIACONTROLTONE** structures
arrCallStates	Array of **LINEMEDIACONTROLCALLSTATE** structures

TSPITONEMONITOR

This structure is used to manage the tone collection process on a single **CTSPICallAppearance** object.

Data Member	Description
dwToneListID	Unique tone identifier for TAPI
arrTones	Array of **LINEMONITORTONE** structures.