TSP++ Version 3.0

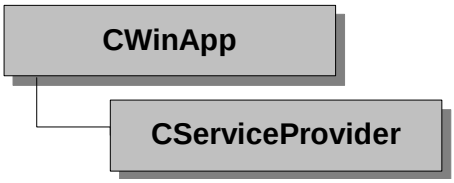
TSP++ User Interface Library Reference Guide

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

TSP+++ Version 3.046

Table of Contents

CServiceProviderUI



In the TSP++ User Interface library (SPLUI), the basic class that will always be present is a class derived from **CServiceProviderUI**. The **CServiceProviderUI** object is the application object that represents the required MFC **CWinApp** object.

The **CServiceProviderUI** object provides a global object where all the configuration information for the service provider is stored (devices, lines, phones, addresses, etc.)

> **Note:** It is important to remember that the user-interface DLL generated with this library is *not* the TSP. The line and phone objects maintained by this library are not pointers to the objects that the TSP manipulates! In many cases the methods are the same and perform similar functions but these objects in the UI library are a subset of the main telephony objects contained in the TSP.

GetUISP

The **CServiceProviderUI** object can be found at any time or place in the UI DLL code by using the inline function **GetUISP**. This is prototyped as:

```
CServiceProviderUI* GetUISP();
```

User Interface Initialization

When any user-interface event occurs, the TAPI Server (**TAPISRV**) will ask the service provider for a user-interface DLL to load into the calling applications address space. This is true even for *spontaneous* dialogs which were initiated by the service provider. This is done by calling the **TSPI_providerUIIdentify**. The default action for the TSP++ library is to return the second parameter from the **CServiceProviderUI** constructor (of the TSP, not this library).

The specified user-interface DLL (which is created using this library) is then loaded by a thread created by **TAPI32.DLL**. It is loaded in the context of the application that is making the UI request (or the application that started the asynchronous request that this UI event is being shown for).

When the TSPUI dynamic link library is loaded into memory, the **CServiceProviderUI** constructor and **CServiceProviderUI::InitInstance** methods will be called.

In the constructor, the various telephony objects should be overridden using the **SetRuntimeObjects** method. This will allow the user-interface to store configuration information in the registry for each defined object type.

Persistent data storage

Any persistent configuration information that will be used by the service provider that is independent of the telephony objects should be stored using the following methods in the library, which have been created for this purpose:

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

These methods are identical in functionality to the methods in the service provider.

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

Spontaneous Dialog Creation

Dialogs can be invoked through two different methods. The first is when an application specifically requests a user-interface event through the normal TAPI functions. These include **lineConfigDialog**, **lineConfigDialogEdit**, **providerConfig**, **phoneConfig**, etc.

The second method is when the service provider itself initiates a user-interface event to inform the user about a specific asynchronous request. An example of this is the Unimodem talk-drop dialog when a voice call is placed. This is called a *spontaneous* dialog event since the application didn't specifically request the UI event.

When a request is made of the user-interface library to show a spontaneous dialog, the **providerGenericDialog** method is invoked. This method should be overridden by the derived UI DLL to instantiate the proper dialog based on the device-specific parameters passed.

Once the dialog has been created, the dialog code should add the created dialog window handle to the UI list using the **AddUIDialog** method and signal the passed *hEvent* handle so that TAPI may pass information to and from the provider using the **TUISPI_providerGenericDialogData** function.

When a data block is sent from the service provider to the user-interface dialog through the **TUISPI_providerGenericDialogData** function, the **providerGenericDialogData** method is invoked. This should be overridden by the UI DLL. The UI DLL can then use

the **FindUIDialog** method to locate the proper dialog based on the passed *htDlgInst* parameter (this was associated with the dialog by the **AddUIDialog** method).

Once the dialog exits, it should call the **RemoveUIDialog** to remove itself from the UI map so that future requests to **FindUIDialog** will not return an invalid window object.

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**).

Each of the derived objects must be declared with the **DECLARE_DYNCREATE** MFC macro in order to be overridden.

The SPLUI library automatically loads all the object information from the registry in the **CWinApp::InitInstance** override (**CServiceProviderUI::InitInstance**). If it detects that there is object information in the registry then it will load it.

The library *does not* automatically save object information into the registry. This is not done specifically so that the derived user interface module has complete control over when to save information (for OK/CANCEL support). If you make changes to the object information in the **CServiceProviderUI** or **CTSPUIDevice** objects and you want that information saved into the registry, you **must** call the **SaveObjects** method.

Constructor and Destructor

CServiceProviderUI	Constructs a **CServiceProviderUI** object. This takes the TSP name that is used to locate configuration information in the registry.
~CServiceProviderUI	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, and address).

Operations - Public Members

AddDevice	Add a new device into the service provider list. This is done automatically in **InitInstance** if there are no devices present.
AddUIDialog	This associates a dialog (or any CWnd pointer) with a TAPI *htDlgInst* parameter so that it may be located later.

DeleteProfile	Delete a profile from the registry.
FindUIDialogFromInstance	This locates a spontaneous dialog based on a *htDlgInst* parameter from TAPI.
GetDevice	Return a pointer to a **CTSPUIDevice** based on the permanent provider ID.
GetDeviceByIndex	Return a pointer to a **CTSPUIDevice** based on an index.
GetDeviceCount	Returns a count of existing devices.
GetPermanentIDFromLineDeviceID	Return a permanent line device identifier from a TAPI line device index.
GetPermanentIDFromPhoneDeviceID	Return a permanent phone device identifier from a TAPI phone device index.
GetProviderIDFromLineDeviceID	Return the permanent provider id from a line device id
GetProviderIDFromPhoneDeviceID	Return the permanent provider id from a phone device id.
GetProviderInfo	Returns provider specific information that was supplied by the constructor.
IsProviderInstalled	Return whether the provider is already installed in the TAPI sub-system. This may be used to ensure that the provider is not installed more than once.
InvokeTSP	Call the TSP driver through the **TSPI_providerGenericDialogData** method.
ReadProfileDWord	Reads a DWORD from the registry section for this provider.
ReadProfileString	Reads a string from the registry section for this provider.
RemoveUIDialog	This removes a dialog pointer from the spontaneous dialog list. It should have been added with the **AddUIDialog** method.
RenameProfile	Rename a profile in the registry section for this provider.
ResetConfiguration	Delete all the device information from the registry for this service provider.
WriteProfileDWord	Writes a DWORD to the registry section for this provider.
WriteProfileString	Writes a string to the registry section for this provider.

Overridables - Public Members

ExitInstance	Called when the service provider is unloaded.
InitInstance	Called when the service provider is first loaded to initialize any internal data.
LoadObjects	This method loads all the object information in from the registry. It is called automatically in the **InitInstance** override if the information is in the registry.
SaveObjects	This method saves the current object states into the registry. It should be called when the user-interface module is unloading, either in the destructor of the **CServiceProviderUI** object or

	the end of a **providerXXX** method. This is *not* done automatically.

Overridables - TAPI Members

providerConfig	This handles the **TUISPI_providerConfig** function. It is called when the provider is being configured by an application or by the telephony control panel applet.
providerGenericDialog	This is called to start a spontaneous user interface dialog.
providerGenericDialogData	This is called to pass information to a created user interface dialog.
providerInstall	This is called to install the provider into the TAPI sub-system.
providerRemove	This is called to remove a provider from the TAPI sub-system.
lineConfigDialog	Called in response to the **TUISPI_lineConfigDialog** function that is used by TAPI to configure a specific line device.
lineConfigDialogEdit	Called in response to the **TUISPI_lineConfigDialogEdit** function that is used by TAPI to configure a specific line device and return the changes in an opaque data structure which may be passed back to the service provider without user-intervention.
phoneConfigDialog	Called in response to the **TUISPI_phoneConfigDialog** function that is used by TAPI to configure a specific phone device.

CServiceProviderUI::AddDevice

unsigned int AddDevice(DWORD *dwProviderID***);**
unsigned int AddDevice(CTSPUIDevice* *pDevice***);**

dwProviderID	The permanent provider ID to add the device for.
pDevice	The created device object to add.

Remarks

This method is used to add a new device into the service provider list. In general there will only be a single device per service provider.

This should be called by the **providerInstall** method if it is determined that the provider may be installed into the telephony system. Either version of the method may be used depending on if the derived **CTSPUIDevice** object has a different constructor.

CServiceProviderUI::AddUIDialog

void AddUIDialog(HTAPIDIALOGINSTANCE *htDlgInst***, CWnd*** *pwnd***);**

htDlgInst	The dialog instance handle passed from TAPI.
pwnd	The window object to associate it with.

Remarks

This method is used associate a TAPI dialog instance handle with a created dialog or CWnd-derived window object. It may then be located using the **FindUIDialog** method at a later point (presumably during a **providerGenericDialogData** call).

CServiceProviderUI::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 **TUISPI_providerRemove**.

Return Value

TRUE if the section was removed.

CServiceProviderUI::GetDevice

CTSPUIDevice* GetDevice (DWORD *dwPermanentProviderID***);**

dwPermandProviderID Permanent Provider ID to lookup.

Remarks

This method returns a pointer to the **CTSPUIDevice** object that was created for the given permanent provider identifier.

Return Value

A pointer to a **CTSPUIDevice** object, NULL if none match the given criteria.

CServiceProviderUI::GetDeviceByIndex

CTSPUIDevice* GetDeviceByIndex(unsigned int *iIndex***);**

iIndex The array position to retrieve the information from.

Remarks

This method returns a pointer to a **CTSPUIDevice** object that is stored at the given array position.

Return Value

A pointer to the device object or NULL if the array index is out of bounds.

CServiceProviderUI::GetDeviceCount

unsigned int GetDeviceCount() const;

Remarks

This method returns the total number of **CTSPUIDevice** objects that have been added to the service provider list (either by an explicit **AddDevice** method or through loading object information from the registry.

Return Value

The number of **CTSPUIDevice** objects contained in the service provider configuration.

CServiceProviderUI::GetPermanentIDFromLineDeviceID

LONG GetPermanentIDIDFromLineDeviceID(DWORD *dwDeviceID***,**
LPDWORD *lpdwPPid***);**

dwDeviceID	A line device ID from TAPI
lpdwPPid	The returning line id for the given line device.

Remarks

This method returns a line provider id from a given line device id. It calls the service provider to perform the translation, so this method can only be used after the service provider has been installed (i.e. not from the **providerInit** method).

Return Value

TAPI result code

CServiceProviderUI::GetPermanentIDFromPhoneDeviceID

LONG GetPermanentIDFromPhoneDeviceID(DWORD *dwDeviceID***,**
LPDWORD *lpdwPPid***);**

dwDeviceID	A phone device ID from TAPI
lpdwPPid	The returning phone id for the given line device.

Remarks

This method returns a permanent phone id from a given phone device id. It calls the service provider to perform the translation, so this method can only be used after the service provider has been installed (i.e. not from the **providerInit** method).

Return Value

TAPI result code

CServiceProviderUI::GetProviderIDFromLineDeviceID

LONG GetProviderIDFromLineDeviceID(DWORD *dwDeviceID***,**
LPDWORD *lpdwPPid***);**

dwDeviceID	A line device ID from TAPI
lpdwPPid	The returning line id for the given line device.

Remarks

This method returns a permanent provider id from a given line device id. It calls the service provider to perform the translation, so this method can only be used after the service provider has been installed (i.e. not from the **providerInit** method).

Return Value

TAPI result code

CServiceProviderUI::GetPermanentIDFromPhoneDeviceID

LONG GetProviderIDFromPhoneDeviceID(DWORD *dwDeviceID***,**
LPDWORD *lpdwPPid***);**

dwDeviceID	A phone device ID from TAPI
lpdwPPid	The returning phone id for the given line device.

Remarks

This method returns a permanent provider id from a given phone device id. It calls the service provider to perform the translation, so this method can only be used after the service provider has been installed (i.e. not from the **providerInit** method).

Return Value

TAPI result code

CServiceProviderUI::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.

CServiceProviderUI::FindUIDialog

CWnd* FindUIDialogFromInstance(HTAPIDIALOGINSTANCE *htDlgInst***);**

htDlgInst	The dialog instance handle passed from TAPI.

Remarks

This method is used to lookup a window object from a TAPI dialog instance handle. The two should have been associated by an earlier call to **AddUIDialog**.

Return Value

The window object matched to the passed dialog instance handle from TAPI. NULL if no window is associated with the given key.

CServiceProviderUI::InvokeTSP

LONG InvokeTSP(LPVOID *lpParams***, DWORD** *dwSize***);**
LONG InvokeTSP(HTAPIDIALOGINSTANCE *htDlgInst***,**
LPVOID *lpParams***, DWORD** *dwSize***);**

lpParams	Parameter block to pass to the TSP
dwSize	Size of the parameter block for RPC transmission.
htDlgInst	A specific dialog instance to inform the TSP about. This is passed to the user-interface library through a **providerGenericDialog** event.

Remarks

This method is used to transmit information from the user interface module directly to the TSP.

Return Value

TAPI return value.

CServiceProviderUI::IsProviderInstalled

LONG IsProviderInstalled(LPCTSTR *pszProviderName***,**
LPDWORD *lpdwPPid***) const;**

pszProviderName	Full name of the TSP (i.e. **JTSP.TSP**)
lpdwPPid	Returning permanent provider id if already installed.

Remarks

This method can be used to test whether the provider has been installed into the TAPI sub-system. The main usage for this would b during the processing of the **providerInit** function to ensure that only one copy of the TSP is ever installed.

Return Value

TAPI return value.

CServiceProviderUI::lineConfigDialog

virtual LONG lineConfigDialog(DWORD *dwDeviceID***, CWnd*** *pwndOwner***,**
CString& *strDeviceClass***);**

dwDeviceID	Line Device ID to configure
pwndOwner	Owner window handle for any dialog created.
strDeviceClass	Specific device class to configure.

Remarks

This method is called when TAPI is configuring a specific line through the **lineConfigDialog** function. It should be overridden if the provider allows individual line configuration adjustment.

Return Value

TAPI return value.

CServiceProviderUI::lineConfigDialogEdit

virtual LONG lineConfigDialogEdit(DWORD *dwDeviceID***,**
CWnd* *pwndOwner***, CString&** *strDeviceClass***,**
const LPVOID *lpDeviceConfigIn***, DWORD** *dwSize***,**
LPVARSTRING *lpDeviceConfigOut***);**

dwDeviceID	Line Device ID to configure
pwndOwner	Owner window handle for any dialog created.
strDeviceClass	Specific device class to configure.
lpDeviceConfigIn	Passed configuration (opaque) information
dwSize	Size of the configuration block
lpDeviceConfigOut	Returning configuration information.

Remarks

This method is called when TAPI is configuring a specific line through the **lineConfigDialogEdit** function. It should be overridden if the provider allows individual line configuration adjustment and can pass back the changes to configuration for the TSP.

Return Value

TAPI return value.

CServiceProviderUI::LoadObjects

virtual void LoadObjects();

Remarks

This method is called to load object configuration information from the registry. This is called automatically by the **InitInstance** method if the information exists in the registry.

CServiceProviderUI::phoneConfigDialog

virtual LONG phoneConfigDialog(DWORD *dwDeviceID***, CWnd*** *pwndOwner***, CString&** *strDeviceClass***);**

dwDeviceID	Phone Device ID to configure
pwndOwner	Owner window handle for any dialog created.
strDeviceClass	Specific device class to configure.

Remarks

This method is called when TAPI is configuring a specific phone through the **phoneConfigDialog** function. It should be overridden if the provider allows individual phone configuration adjustment.

Return Value

TAPI return value.

CServiceProviderUI::providerConfig

virtual LONG providerConfig(DWORD *dwPPID***, CWnd*** *pwndOwner***);**

dwPPID	Permanent provider id to configure.
pwndOwner	Owner window handle for any dialog created.

Remarks

This method is called when TAPI is configuring the service provider from the control panel applet or from an application calling the **providerConfig** function. It should always be overridden for provider configuration.

Return Value

TAPI return value.

CServiceProviderUI::providerGenericDialog

virtual LONG providerGenericDialog (HTAPIDIALOGINSTANCE *htDlgInst***, LPVOID** *lpParams***, DWORD** *dwSize***, HANDLE** *hEvent***);**

htDlgInst	Dialog instance handle which may be used as the first parameter to the **InvokeTSP** method when talking to the TSP about this specific dialog.
lpParams	Parameters passed from the TSP for this dialog.
dwSize	Size of the parameter block
hEvent	Event handle which should be signaled when the dialog is actually created (in the **OnInitDialog** handler of the created dialog).

Remarks

This method is called when the TSP initiates a *spontaneous* dialog (i.e. a dialog that was not directly invoked by an application) for a running asynchronous request. An example of this might be a talk-drop dialog (such as displayed by Unimodem).

This method should be overridden to support spontaneous dialogs from the TSP.

The *hEvent* parameter is a Win32 handle to an event object created by TAPI. This event should be signaled by the UI DLL through Win32 **SetEvent** function when the UI DLL has completed initialization of this dialog box and is prepared to receive additional dialog box data through the **TUISPI_providerGenericDialogData** function. Data sent by the associated service provider for this dialog box is blocked by TAPI until the UI DLL signals this event, giving the dialog the opportunity to perform any necessary initialization.

The UI DLL should signal the event as quickly a possible to avoid blocking calls to **TUISPI_providerGenericDialogData**.

Once the dialog is created, it may associate it's window object with the *htDlgInst* parameter using the **AddUIDialog** method. This will allow the window object to be located if the service provider passes information to it at a later point using the **providerGenericDialogData** method.

The user-interface dialog may pass data back to the service provider using the **InvokeTSP** method and passing the *htDlgInst* parameter as the first parameter.

Note that as of this writing, TAPI 2.1 does not support spontaneous dialogs off the local service provider machine (i.e. dialogs are not created on machines using **REMOTETSP**).

Return Value

TAPI return value.

CServiceProviderUI::providerGenericDialogData

virtual LONG providerGenericDialogData(
HTAPIDIALOGINSTANCE *htDlgInst***, LPVOID** *lpParams***,**
DWORD *dwSize***);**

htDlgInst	TAPI Dialog instance handle for the UI dialog.
lpParams	Parameters passed from the TSP for this dialog.
dwSize	Size of the parameter block

Remarks

This method is called when the TSP sends information to a created spontaneous dialog. It will be invoked if the TSP uses the **SendDialogInstanceData** method of the line object.

This method should be overridden to reception of data from the TSP.

If the user-interface dialog associated the *htDlgInst* with itself during its initialization, then the **FindUIDialog** method can be used to locate the proper CWnd-derived object to send the data to.

This method will not be called until the *hEvent* of the **providerGenericDialog** method is signaled (see the above remarks for **providerGenericDialog**).

Return Value

TAPI return value.

CServiceProviderUI::providerInstall

virtual LONG providerInstall(DWORD *dwPermanentProviderID***,**
CWnd* *pwndOwner***);**

dwPermanentProviderID	Permanent provider ID assigned by TAPI
pwndOwner	Window to use as the dialog owner for any user-interface created in response to this.

Remarks

This method is called when TAPI is installing the TSP into the TAPI sub-system.

It should perform any checks necessary for the TSP installation (whether it is already installed, hardware connected, etc.) and prompt for configuration data if it is necessary for the TSP to run.

The derived object should always pass control to the default implementation in the class library ***after*** it has successfully installed the service provider. If the provider is not to be installed *do not* pass control to the default implementation as it writes installation registry key information.

Return Value

TAPI return value.

CServiceProviderUI::providerRemove

virtual LONG providerRemove(DWORD *dwPermanentProviderID***,**
CWnd* *pwndOwner***);**

dwPermanentProviderID	Permanent provider ID assigned by TAPI
pwndOwner	Window to use as the dialog owner for any user-interface created in response to this.

Remarks

This method is called when TAPI is removing the TSP from the TAPI sub-system. The default implementation should always be called in order to remove registry keys.

Return Value

TAPI return value.

CServiceProviderUI::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.

CServiceProviderUI::ReadProfileString

CString 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 **TUISPI_providerRemove**.

Return Value

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

CServiceProviderUI::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 **TUISPI_providerRemove**.

Return Value

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

CServiceProviderUI::ResetConfiguration

void ResetConfiguration();

Remarks

This method removes all the existing configuration from the internal structures. It may be used to quickly remove all objects so that the user-interface may re-create all the devices before saving the objects out.

CServiceProviderUI::SaveObjects

void SaveObjects();

Remarks

This method saves the internal object structures into the registry in object format. This will allow the TSP (and future invocations of the UI DLL) to reload the information and recreate the objects automatically.

CServiceProviderUI::SetRuntimeObjects

Protected

void SetRuntimeObjects(CRuntimeClass* *pDevObj***,**
CRuntimeClass* *pLineObj* **= NULL, CRuntimeClass*** *pAddrObj* **= NULL,**
CRuntimeClass* pPhoneObj **= NULL);**

pDevObject	This is the class object that should be used in place of the standard device object. It must be derived from the **CTSPUIDevice** object.
pLineObject	This is the class object that should be used in place of the standard line device object. It must be derived from the **CTSPUILineConnection** object.
pAddrObject	This is the class object that should be used in place of the standard address object. It must be derived from the **CTSPUIAddressInfo** object.
pPhoneObject	This is the class object that should be used in place of the standard phone device object. It must be derived from the **CTSPUIPhoneConnection** 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 **CServiceProviderUI** object to override each desired class. If the method passed NULL in for any parameter, the default object type is used.

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.

CServiceProviderUI::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 **TUISPI_providerRemove**.

Return Value

TRUE if the value was stored successfully.

CServiceProviderUI::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

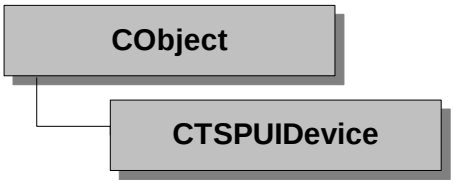
This method writes a string into the storage section devoted to the service provider inside the registry. The provider ID is used to distinguish between multiple devices within the provider. Since TAPI guarantees that they will be unique system-wide, they are used as part of the key to store the data.

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

Return Value

TRUE if the value was stored successfully.

CTSPUIDevice



The **CTSPUIDevice** object provides a mini-object for storing configuration information related to the connection from the TSP to the physical hardware device. The object is also the storage place for maintaining line and phone information related to a single hardware device (such as a PBX).

It is recommended that this object be overridden to store information related to connections and timeouts for a device. This could be things such as TCP/IP information, COMM port information, port driver names, etc.

Device Initialization

When the user interface DLL is first loaded, no devices are present. If the object configuration information is stored in the registry, then it is reloaded and setup during the **InitInstance** method of **CServiceProviderUI**.

If no object information exists in the registry (probably due to this being a **providerInstall** request) then the derived UI DLL should call the **CServiceProviderUI::AddDevice** method to create/add a device object to the service provider list.

Once the device is added, line, phone, and agent information may be added to the configuration.

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.

Each line and phone should be added to the configuration using the **AddLine** and **AddPhone** methods. If the object information has been stored in the registry, then this information is reloaded when the UI DLL is loaded.

Agent Information

The device object also stores the agent information, this includes agent activities and groups. Note that agents themselves are not stored by TSP++ since the data structure is not tracked by TAPI.

Each agent and group should be added to the configuration using the **AddAgentActivity** and **AddAgentGroup** methods.

Serialization

The data contained within the device object is serialized to the registry when the **CServiceProviderUI::SaveObjects** method is invoked. The information stored in the registry is:

1. Permanent Provider ID

Additional information may be added to this by overriding the **read** and **write** methods in the object. If new information is appended to the registry stream then the receiving object in the TSP must be programmed to retrieve the information from the stream in the same order as it is saved. This is extremely important as other objects append their data to the same registry stream.

Constructor and Destructor

CTSPUIDevice	Constructs a **CTSPUIDevice** object.
~CTSPUIDevice	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.
AddLine	Adds a new line to the device.
AddPhone	Adds a new phone to the device
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.
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
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.
RemoveAgentActivity	Removes an agent activity from the device.
RemoveAgentGroup	Removes an agent group from the device.
RemoveLine	Remove an existing line from the device.
RemovePhone	Remove an existing phone from the device.

Overridables - Protected Members

AllocStream	This function is used to allocate the stream object to save and restore provider information.
read	Reads configuration information about the device from the registry using a registry iostream.
write	Writes configuration information into the registry.

CTSPUIDevice::AddAgentActivity

unsigned int AddAgentActivity(DWORD *dwID***, LPCTSTR** *pszName***);**
unsigned int AddAgentActivity (TAgentActivity* *pAct***);**

dwID	Unique agent activity identifier.
pszName	Text name of the activity.
pAct	Created agent activity structure.

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.

CTSPUIDevice::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.

CTSPUIDevice::AddLine

unsigned int AddLine(CTSPUILineConnection* *pLine***);**

pLine	Line object to add to the device.

Remarks

This method adds a new line object to the device. The object should have already been created and should not be destroyed directly by the derived code as it is now managed by the SPLUI library.

Return Value

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

CTSPUIDevice::AddPhone

unsigned int AddPhone(CTSPUIPhoneConnection* *pPhone***);**

pPhone Phone object to add to the device.

Remarks

This method adds a new phone object to the device. The object should have already been created and should not be destroyed directly by the derived code as it is now managed by the SPLUI library.

Return Value

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

CTSPUIDevice::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 **LoadObjects** and **SaveObjects** functions. 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 save and load information for the provider.

CTSPUIDevice::CTSPUIDevice

CTSPUIDevice(DWORD *dwPermProviderID***);**

dwPermProviderID Provider ID assigned by TAPI.

Remarks

This is the constructor for the device object.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::FindLineConnectionByPermanentID

CTSPUILineConnection* 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 **CTSPUILineConnection** 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.

CTSPUIDevice::FindPhoneConnectionByPermanentID

CTSPUIPhoneConnection* 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 **CTSPUIPhoneConnection** 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.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::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.

CTSPUIDevice::GetLineConnectionInfo

CTSPUILineConnection* 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 **CTSPUILineConnection** 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.

CTSPUIDevice::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.

CTSPUIDevice::GetPhoneConnectionInfo

CTSPUIPhoneConnection* 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 **CTSPUIPhoneConnection** 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.

CTSPUIDevice::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.

CTSPUIDevice::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 **CServiceProviderUI** 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.

CTSPUIDevice::read

protected

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.

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.

CTSPUIDevice::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

CTSPUIDevice::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

CTSPUIDevice::RemoveLine

void RemoveLine(CTSPUILineConnection* *pLine***);**
void RemoveLine(unsigned int *iLine***);**

pLine	Line device object to remove from the system.
iLine	Numerical index of the line to remove.

Remarks

This method removes a line from the configuration. The line object will not be saved when the next **CServiceProviderUI::SaveObjects** is called.

The line object itself is not deleted by these methods.

CTSPUIDevice::RemovePhone

void RemovePhone(CTSPUIPhoneConnection* *pPhone***);**
void RemovePhone(unsigned int *iPhone***);**

pPhone	Phone device object to remove from the system.

iPhone Numerical index of the phone to remove.

Remarks

This method dynamically removes a phone from the configuration. The phone object will not be saved when the next **CServiceProviderUI::SaveObjects** is called.

The phone object itself is not deleted by these methods.

CTSPUIDevice::ResetConfiguration

void ResetConfiguration();

Remarks

This method removes all the existing configuration from the internal structures. It may be used to quickly remove all line and phone objects so that the user-interface may re-create all the devices before saving the objects out.

CTSPUIDevice::write

protected

virtual std::ostream& write(std::ostream& *ostm***);**

ostm output iostream to write information into.

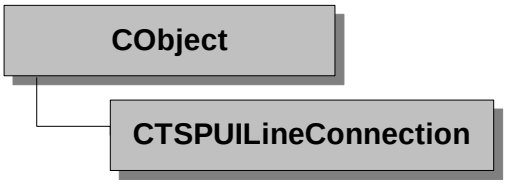
Remarks

This method is called when the object information is being saved out into the registry.

It may be overridden to write additional information into 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.

CTSPUILineConnection



The **CTSPUILineConnection** object provides a mini-object for storing configuration information related to a single line device within the telephony hardware.

Address Objects

Channels on the line are represented by **CTSPUIAddressInfo** objects owned by the line. They are created using the **CreateAddress** or **AddAddress** methods and may be enumerated using the **GetAddressCount** and **GetAddress** methods.

Each line should have at least one address object in order to make or receive calls.

Serialization

The data contained within the line object is serialized to the registry when the **CServiceProviderUI::SaveObjects** method is invoked. The information stored in the registry is:

1. Permanent Line device id
2. Line Type
3. Related Phone id
4. Line Name
5. Addresses on the line
6. Whether the line supports agent features.

Additional information may be added to this by overriding the **read** and **write** methods in the object. If new information is appended to the registry stream then the receiving object in the TSP must be programmed to retrieve the information from the stream in the same order as it is saved. This is extremely important as other objects append their data to the same registry stream.

Constructor and Destructor

CTSPUILineConnection	Constructs a **CTSPUILineConnection** object.
~CTSPUILineConnection	Destructor that deallocates all the information in the line object.

Operations - Public Methods

AddAddress	Add a new **CTSPUIAddressInfo** object to this line.
CreateAddress	Create and add a new **CTSPUIAddressInfo** object to this line.
EnableAgentSupport	Turn the agent support on or off for this line.
GetAddress	Retrieve a specific **CTSPUIAddressInfo** associated with this line.
GetAddressCount	Get the total number of addresses on the line.
GetAssociatedPhone	Get the associated phone device for this line.
GetDeviceInfo	Get the owning **CTSPUIDevice** pointer.
GetLineType	Get the line type.
GetName	Get the textual line name.
GetPermanentDeviceID	Get the permanent line device identifier.
RemoveAddress	Remove a specific **CTSPUIAddressInfo** object from this line.
SetAssociatedPhone	Associate a line and a phone device together.
SetLineType	Set the line type.
SetName	Set the textual line name.
SetPermanentDeviceID	Set the permanent line device identifier.
SetProtocolCLSID	TAPI 3.0. This sets the default protocol CLSID for the line.
SetMSPGUID	TAPI 3.0. This sets the GUID for the associated media service provider.
SupportsAgents	Return whether the line supports agents.

Overridables - Protected Methods

read	Reads configuration information about the line from the registry using a registry iostream.
write	Writes configuration information into the registry.

CTSPUILineConnection::AddAddress

unsigned int AddAddress(CTSPUIAddressInfo* *pAddr***);**

pAddr	The address object to add to this line.

Remarks

This method adds the created address object to the line device's managing array. This address will now be considered a channel for this line device and be saved in the line device configuration in the registry.

Return Value

The index position where the address was added.

CTSPUILineConnection::CreateAddress

unsigned int 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	**TAPI 3.0 LINEADDRESSTYPE_xxx** constant.

Remarks

This method is used to create a new address on a line. 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

The index position where the address was added.

CTSPUILineConnection::CTSPUILineConnection

CTSPUILineConnection(DWORD *dwDeviceID***, int** *iType***, LPCTSTR** *pszName***);**

dwDeviceID	Permanent line device id
iType	Line Type (see **GetLineType**)
pszName	Textual line name

Remarks

This is the constructor for the line connection object.

CTSPUILineConnection::~CTSPUILineConnection

~CTSPUILineConnection()

Remarks

This is the destructor for the line connection object.

CTSPUILineConnection::EnableAgentSupport

void EnableAgentSupport(bool *fEnable***);**

fEnable	Whether to enable/disable agent support

Remarks

This method turns agent support on and off for the given line device.

When the line device is loaded into the TSP data structures, the **EnableAgentProxy** method will be called if agent support is enabled.

CTSPUILineConnection::GetAddress

CTSPUIAddressInfo* GetAddress (unsigned int *iAddressID***) const;**
CTSPUIAddressInfo* GetAddress (LPCTSTR *lpszDialableAddr***) const;**

iAddressID	Numeric address id (zero-based) to locate.
lpszDialableAddr	Dialable number to locate

Remarks

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

Return Value

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

CTSPUILineConnection::GetAddressCount

unsigned int 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.

CTSPUILineConnection::GetAssociatedPhone

CTSPUIPhoneConnection* GetAssociatedPhone() const;

Remarks

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

Return Value

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

CTSPUILineConnection::GetDeviceInfo

CTSPIDevice* GetDeviceInfo() const;

Remarks

This method returns the device parent for this line object.

Return Value

Parent **CTSPUIDevice** object for the line or NULL if the line has not been associated with a device object yet.

CTSPUILineConnection::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 **CTSPUILineConnection**).

CTSPUILineConnection::GetName

LPCTSTR GetName() const;

Remarks

This method returns the name that has been assigned to the line device either by the constructor or the **SetName** method. This is reported in the **LINEDEVCAPS** by the full TSP object.

Return Value

Textual name of the line device.

CTSPUILineConnection::GetPermanentDeviceID

DWORD GetPermanentDeviceID() const;

Remarks

This method returns the permanent line device ID that has been assigned to this line. It should be a unique value across all the lines on this device.

Return Value

32-bit permanent line device id.

CTSPUILineConnection::read

protected

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.

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.

CTSPUILineConnection::RemoveAddress

void RemoveAddress(unsigned int *iAddressID***);**

iAddressID The zero-based index for the address to remove from this line.

Remarks

This method is called to remove an address object from the current line device. This does *not* delete the given address object, it simply de-associates it from the line device owner.

CTSPUILineConnection::SetAssociatedPhone

void SetAssociatedPhone(DWORD *dwPhoneID***);**

dwPhoneID The permanent phone identifier to associate with this line.

Remarks

This method associates a phone device with the given line device object. The two are considered the *same* physical device after this association and will be tied together in the TSP code when it reads the configuration information.

Return Value

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

CTSPUILineConnection::SetLineType

void SetLineType(int *iLineType***);**

iLineType The type of line this object represents.

Remarks

This method sets the line type for this line device. This can be one of the following:

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

all of these are enumerated constant of the **CTSPUILineConnection** object, so they must be prefaced with the proper scope (i.e. **CTSPUILineConnection::Station**).

CTSPUILineConnection::SetName

void SetName(LPCTSTR *pszName***);**

pszName Name to give this line device

Remarks

This method sets the textual name for the given line device. This is reported in the **LINEDEVCAPS** by the full TSP object.

CTSPUILineConnection::SetPermanentDeviceID

void SetPermanentDeviceID(DWORD *dwDeviceID***);**

dwDeviceID Device ID to assign to this line

Remarks

This method sets the permanent line device ID for this line device. It will be assigned to the full TSP object as the **LINEDEVCAPS. dwPermanentLineID**.

CTSPUILineConnection::SetProtocolCLSID

void SetProtocolCLSID(const GUID& *clsid***);**

clsid TAPI protocol CLSID

Remarks

This method sets the protocol CLSID for this line device. It will be assigned to the full TSP object as the **LINEDEVCAPS.ProtocolGuid**. The default behavior is to set the protocol to PSTN. This function requires TAPI 3.0 negotiation.

CTSPUILineConnection::SetMSPGUID

void SetMSPGUID(const GUID& *guid***);**

guid Unique GUID of the MSP for this line

Remarks

This method sets the COM GUID of the Media Service Provider which will perform media services for this line. This will be reflected in the full TSP object when TAPI calls the **TSPI_lineMSPIdentify** function. This function requires TAPI 3.0 negotiation.

CTSPUILineConnection::SupportsAgents

bool SupportsAgents() const;

Remarks

This method can be used to determine if the given line supports agents. This returns the flag that is set by the **EnableAgentSupport** method.

Return Value

TRUE if the line supports agents, FALSE if it does not.

CTSPUILineConnection::write

protected

virtual std::ostream& write(std::ostream& *ostm***);**

ostm output iostream to write information into.

Remarks

This method is called when the object information is being saved out into the registry.

It may be overridden to write additional information into 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.

CTSPUIPhoneConnection

The **CTSPUIPhoneConnection** object provides a mini-object for storing configuration information related to a single phone device within the telephony hardware.

Phone Objects

The phone contains many different elements such as buttons, display information, lamps, hookswitch information, etc. The UI library supports all these elements in the configuration, but does so as a write-only setting.

It is assumed that phones are static devices that don't change once they are added (i.e. the display will ALWAYS be 40x2 or whatever, there will always be a handset hookswitch, etc.) If the device does change then the phone object must be deleted and then re-created in order to change the object.

Serialization

The data contained within the phone object is serialized to the registry when the **CServiceProviderUI::SaveObjects** method is invoked. The information stored in the registry is:

1. Permanent phone device ID.
2. Associated line device ID.
3. Phone name.
4. Display information (size and row terminator).
5. Button information.
6. Hookswitch information.
7. Download buffer information.
8. Upload buffer information.

Additional information may be added to this by overriding the **read** and **write** methods in the object. If new information is appended to the registry stream then the receiving object in the TSP must be programmed to retrieve the information from the stream in the same order as it is saved. This is extremely important as other objects append their data to the same registry stream.

Constructor and Destructor

CTSPUIPhoneConnection	Constructs a **CTSPUIPhoneConnection** object.
~CTSPUIPhoneConnection	Destructor that deallocates all the information in the phone object.

Operations - Public Methods

AddButton	Add a single button to the phone.
AddDownloadBuffer	Add a single download buffer to the phone
AddHookswitchDevice	Add a single hookswitch device to the phone
AddUploadBuffer	Add a single upload buffer to the phone
GetAssociatedLine	Get the associated line device for this line.
GetDeviceInfo	Get the owning **CTSPUIDevice** pointer.
GetName	Get the textual phone name.
GetPermanentDeviceID	Get the permanent phone device identifier.
SetAssociatedLine	Associate a phone and a line device together.
SetName	Set the textual phone name.
SetPermanentDeviceID	Set the permanent phone device identifier.
SetupDisplay	Setup the characteristics of the phone display.

Overridables - Protected Methods

read	Reads configuration information about the phone from the registry using a registry iostream.
write	Writes configuration information into the registry.

CTSPUIPhoneConnection::AddButton

int AddButton (DWORD *dwFunction***, DWORD** *dwMode***,**
DWORD *dwLampStates***, LPCTSTR** *lpszText***);**

dwFunction	Button function (**PHONEBUTTONFUNCTION_xxx**).
dwMode	Button mode (**PHONEBUTTONMODE_xxx**).
dwLampStates	Available lamp states (**PHONELAMPMODE_xxx**).
lpszText	ASCII Text for button.

Remarks

This method adds a button to the phone model presented to TAPI. Each button added to the phone will be reported through the **PHONEDEVCAPS** and TAPI button information functions in the full TSP object.

Return Value

The position of the button within the internal button array.

CTSPUIPhoneConnection::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. Each buffer added to the phone will be reported through the **PHONEDEVCAPS** structure in the full TSP object.

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.

CTSPUIPhoneConnection::AddHookSwitchDevice

int AddHookSwitchDevice (DWORD *dwHookSwitchDev***,**
DWORD *dwAvailModes***, 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. Each hookswitch added to the phone will be reported through the **PHONEDEVCAPS** structure in the full TSP object.

Return Value

The position of the buffer within the internal tracking array.

CTSPUIPhoneConnection::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. Each buffer added to the phone will be reported through the **PHONEDEVCAPS** structure in the full TSP object.

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.

CTSPUIPhoneConnection::CTSPUIPhoneConnection

CTSPUIPhoneConnection(DWORD *dwDeviceID***, LPCTSTR** *pszName***);**

dwDeviceID	Permanent line device id
pszName	Textual line name

Remarks

This is the constructor for the phone connection object.

CTSPUIPhoneConnection::~CTSPUIPhoneConnection

~CTSPUIPhoneConnection()

Remarks

This is the destructor for the phone connection object.

CTSPUIPhoneConnection::GetAssociatedLine

CTSPUILineConnection* GetAssociatedPhone() const;

Remarks

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

Return Value

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

CTSPUIPhoneConnection::GetDeviceInfo

CTSPIDevice* GetDeviceInfo() const;

Remarks

This method returns the device parent for this phone object.

Return Value

Parent **CTSPUIDevice** object for the phone or NULL if the phone has not been associated with a device object yet.

CTSPUIPhoneConnection::GetName

LPCTSTR GetName() const;

Remarks

This method returns the name that has been assigned to the phone device either by the constructor or the **SetName** method. This is reported in the **PHONECAPS** by the full TSP object.

Return Value

Textual name of the phone device.

CTSPUIPhoneConnection::GetPermanentDeviceID

DWORD GetPermanentDeviceID() const;

Remarks

This method returns the permanent phone device ID which has been assigned to this line. It should be a unique value across all the phones on this device.

Return Value

32-bit permanent line device id.

CTSPUIPhoneConnection::read

protected

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.

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.

CTSPUIPhoneConnection::SetAssociatedLine

void SetAssociatedLine(DWORD *dwLineID***);**

dwLineID Permanent line device ID to associate to this phone.

Remarks

This method associates a phone device with the given line device identifier. The two are considered the *same* physical device after this association and will be tied together in the TSP code when it reads the configuration information.

CTSPUIPhoneConnection::SetName

void SetName(LPCTSTR *pszName***);**

pszName	Name to give this phone device

Remarks

This method sets the textual name for the given phone device. This is reported in the **PHONECAPS** by the full TSP object.

CTSPUIPhoneConnection::SetPermanentDeviceID

void SetPermanentDeviceID(DWORD *dwDeviceID***);**

dwDeviceID	Device ID to assign to this phone

Remarks

This method sets the permanent phone device ID for this line device. It will be assigned to the full TSP object as the **PHONECAPS. dwPermanentPhoneID**.

CTSPUIPhoneConnection::SetupDisplay

void SetupDisplay (int *iColumns***, int** *iRows***, char** *cChar***='\n');**

iColumns	Number of columns in the display.
iRows	Number of rows in the display
cChar	The character which will be used by the service provider to separate rows if not padded out to the full width.

Remarks

This method is used to setup the display configuration for the phone device. The phone only supports a single display unit of a known size. The character separator is used by the TSP++ library to separate rows from each other. If one of the given characters is found in the display it will be assumed that the end-of-row has been found even if the total row length has not been reached.

CTSPUIPhoneConnection::write

protected

virtual std::ostream& write(std::ostream& *ostm***);**

ostm	output iostream to write information into.

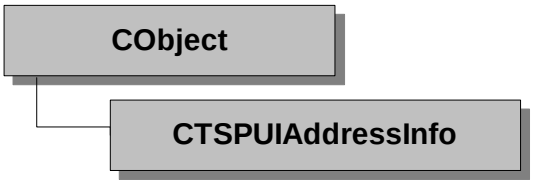
Remarks

This method is called when the object information is being saved out into the registry.

It may be overridden to write additional information into 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.

CTSPUIAddressInfo



The **CTSPUIAddressInfo** object provides a mini-object for storing configuration information related to a single address, or channel on a line.

The address objects are owned by a single line device and are created using the **CTSPUILineConnection::CreateAddress** or directly through a constructor and then added to the line using the **CTSPUILineConnection::AddAddress** method.

Address Components

The address object contains many different data components such as maximum call counts, data rate, dialing information, etc. The UI library supports all these elements in the configuration, but does so as a write-only setting.

It is assumed that addresses are static devices that don't change once it has been created. If the address does change then the address object must be deleted and re-created in order to change the settings.

Serialization

The data contained within the address object is serialized to the registry when the **CServiceProviderUI::SaveObjects** method is invoked. It is saved as part of the line configuration. The information stored in the registry is:

1. Dialable Number
2. Address Name
3. Whether the address supports incoming call traffic.
4. Whether the address supports outgoing call traffic.
5. Media modes supported
6. Bearer mode supported
7. Minimum/Maximum data rates on the address
8. Maximum number of active calls on the address.
9. Maximum number of held calls on the address.
10. Dialing parameters (pause, duration, etc.)

Additional information may be added to this by overriding the **read** and **write** methods in the object. If new information is appended to the registry stream then the receiving object in the TSP must be programmed to retrieve the information from the stream in

the same order as it is saved. This is extremely important as other objects append their data to the same registry stream.

Constructor and Destructor

CTSPUIAddressInfo	Constructs a **CTSPUIAddressInfo** object.
~CTSPUIAddressInfo	Destructor that deallocates all the information in the address object.

Operations - Public Methods

CanAnswerCalls	Returns whether the address is capable of incoming calls.
CanMakeCalls	Returns whether the address can place calls
GetAvailableMediaModes	Returns the supported media modes
GetBearerMode	Returns the supported bearer mode
GetDialableAddress	Returns the dialable number for this address.
GetDialParams	Returns a pointer to the **LINEDIALPARAMS** for this address.
GetMinimumDataRate	Returns the minimum data rate.
GetMaximumDataRate	Returns the maximum data rate.
GetMaxNumActiveCalls	Returns the maximum number of active calls.
GetMaxNumInConference	Returns the maximum number of calls in a conference.
GetMaxNumInTransfConference	Returns the maximum number of calls that can be transferred into a conference.
GetMaxNumOnHoldCalls	Returns the maximum number of calls on Hold at any given time.
GetMaxNumOnHoldPendCalls	Returns the maximum number of calls that can be on hold pending a transfer or conference.
GetName	Returns the textual name for the address.
SetDialableAddress	Allows the dialable address to be modified.
SetName	Allows the textual name to be modified.

Overridables - Protected Methods

Init	Post constructor initialization used when the address is created through a line device.
read	Reads configuration information about the address from the registry using a registry iostream.
write	Writes configuration information into the registry.

CTSPUIAddressInfo::CanAnswerCalls

bool CanAnswerCalls() const;

Remarks

This method returns the value given to the **CTSPUIAddressInfo::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.

CTSPUIAddressInfo::CanMakeCalls

bool CanMakeCalls() const;

Remarks

This method returns the value given to the **CTSPUIAddressInfo::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.

CTSPUIAddressInfo::CTSPUIAddressInfo

CTSPUIAddressInfo(const CTSPUIAddressInfo& *addr***);**
CTSPUIAddressInfo(LPCTSTR *lpszDialableAddr***, LPCTSTR** *lpszAddrName***,**
bool *fAllowIncoming***, bool** *fAllowOutgoing***,**
DWORD *dwAvailMediaModes***, DWORD** *dwBearerMode***,**
DWORD *dwMinRate***, DWORD** *dwMaxRate***,**
LPLINEDIALPARAMS *lpDialParams***, DWORD** *dwMaxNumActiveCalls***,**
DWORD *dwMaxNumOnHoldCalls***,**
DWORD *dwMaxNumOnHoldPendCalls***, DWORD** *dwMaxNumConference***,**
DWORD *dwMaxNumTransConf***);**

addr	Existing address object to copy.
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.

Remarks

This is the constructor for the address object.

CTSPUIAddressInfo::~CTSPUIAddressInfo

virtual ~CTSPUIAddressInfo();

Remarks

This is the destructor for the address object. It may be overridden to delete any additional data added to the object.

CTSPUIAddressInfo::GetAvailableMediaModes

DWORD GetAvailableMediaModes () const;

Remarks

This method is a method to return the **LINEADDRESSCAPS.dwAvailableMediaModes** value from the full TSP object. This was set by the creation of the address object through the media modes passed into the **CTSPUILineConnection::CreateAddress** method.

Return Value

The available media modes supported by the address.

CTSPUIAddressInfo::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 **CTSPUILineConnection::CreateAddress** method.

Return Value

The bearer mode of the address object.

CTSPUIAddressInfo::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 **CTSPUILineConnection::CreateAddress** method.

Return Value

The dialable address of the address object.

CTSPUIAddressInfo::GetDialParams

LPLINEDIALPARAMS GetDialParams();

Remarks

This method returns the default dialing parameters used for this address. The min/max values for all addresses are used to determine the line dialing parameters.

Return Value

Pointer to the dialing parameters for this address object.

CTSPUIAddressInfo::GetMinimumDataRate

DWORD GetMinimumDataRate() const;

Remarks

This method returns the minimum data rate for the address. This is normally used in ISDN or modem data transfers. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The minimum data rate for the address.

CTSPUIAddressInfo::GetMaximumDataRate

DWORD GetMaximumDataRate() const;

Remarks

This method returns the maximum data rate available on the address. This is normally used in ISDN or modem data transfers. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum data rate for the address

CTSPUIAddressInfo::GetMaxNumActiveCalls

DWORD GetMaxNumActiveCalls() const;

Remarks

This method returns the maximum number of active calls on the address. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum number of calls that can be active simultaneously on the address. This does not count conferenced calls.

CTSPUIAddressInfo::GetMaxNumInConference

DWORD GetMaxNumInConference() const;

Remarks

This method returns the maximum number of calls that can be in a single conference on the address. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum number of calls that can be active in a conference.

CTSPUIAddressInfo::GetMaxNumInTransfConference

DWORD GetMaxNumInTransfConference() const;

Remarks

This method returns the maximum number of calls that transferred into a conference simultaneously. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum number of calls that can be transferred into a conference.

CTSPUIAddressInfo::GetMaxNumOnHoldCalls

DWORD GetMaxNumOnHoldCalls() const;

Remarks

This method returns the maximum number of calls that can be holding simultaneously. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum number of calls that can be put on hold at the same time.

CTSPUIAddressInfo::GetMaxNumOnHoldPendCalls

DWORD GetMaxNumOnHoldPendCalls() const;

Remarks

This method returns the maximum number of calls that can be holding for a conference or transfer event simultaneously. This is reported back in the **LINEADDRESSCAPS** structure for the full TSP object.

Return Value

The maximum number of calls that can be put on hold in order to create a conference or to transfer them.

CTSPUIAddressInfo::GetName

LPCTSTR GetName() const;

Remarks

This method returns the name that has been assigned to the address either by the constructor or the **SetName** method. This is reported in the **LINEADDRESSCAPS** by the full TSP object.

Return Value

Textual name of the address on the line.

CTSPUIAddressInfo::Init

Protected

virtual void Init (CTSPUILineConnection* *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 *dwAddressType* = **0);**

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.
dwAddressType	**TAPI 3.0 LINEADDRESSTYPE_xxx** constant.

Remarks

This method is used to initialize an address object. It is called directly after the constructor of the **CTSPUIAddressInfo** object in response to a **CTSPUILineConnection::CreateAddress** call.

CTSPUIAddressInfo::read

protected

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 address object information is contained within 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.

CTSPUIAddressInfo::SetDialableAddress

void SetDialableAddress(LPCTSTR *pszAddress***);**

pszAddress New dialable address for this object.

Remarks

This method changes the dialable address of the address object. This method changes then reported **LINEADDRESSCAPS.dwDialableAddr** fields in the full TSP object.

CTSPUIAddressInfo::SetName

void SetName(LPCTSTR *pszName***);**

pszName Name to give this address on the line.

Remarks

This method sets the textual name for the given address. This is reported in the **LINEADDRESSCAPS** by the full TSP object.

CTSPUIAddressInfo::write

protected

virtual std::ostream& write(std::ostream& *ostm***);**

ostm output iostream to write information into.

Remarks

This method is called when the object information is being saved out into the registry.

It may be overridden to write additional information into 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.