SwyxIt! Client SDK

Swyx Solutions GmbH
Emil-Figge-Str. 86
44227 Dortmund

1 Overview

One strength of SwyxIt! is the possibility of integration into 3rd party software. This allows starting calls from CRM software or popping up screens depending on the caller id. It is even possible to build systems for predictive dialing. SwyxIt! provides two ways of integration into 3rd party applications: TAPI and Client SDK. Both interfaces allow at least basic call control. If there are no special reasons for using TAPI (e.g. an existing TAPI application), we recommend to use the Client SDK. Using Client SDK you may use any call control options of SwyxIt!.
The SDK contains samples in C++, C# and VB. When you have read this documentation, please pay some attention to the sample code, showing some common tasks.

1.1 Disclaimer

You may use the Client SDK on your own risk. The SDK technically allows using the product in a way that is not supported or intended. By doing this some functions might not work as expected.

This Client SDK and the sample code are provided as is. The SDK and the documented API is subject to change without notice. Further development of the product might require changes of the API that can lead to incompatibility with previous versions. We don't guarantee that all currently supported functions are part of future versions. Nevertheless the API is used internally as well and we will not change it without very good reasons.

The Line Manager COM interface has been tested internally using…

- C++ using Visual Studio .NET 2010
- Visual Basic .NET using Visual Studio .NET 2010
- C# using Visual Studio .NET 2010

We don't guarantee that the API works with any other versions of Microsoft Visual Studio.

The Line Manager COM API should be usable from any programming language or tool that can interoperate with COM. But we cannot guarantee that the API works with other programming languages or tools from other vendors like Borland Delphi.

1.2 Operation Mode

1.2.1 Normal Mode vs. Power Dial Mode

- Typically SwyxIt! runs in normal operation mode. In normal operation mode the audio (voice, dial tone, alerting, busy…) of the selected line (the line in focus) is linked to a local sound device (handset, headset, speakers…). There is always exactly one line selected. When a line is hooked off (e.g. because the user has clicked on it), it becomes the selected line and the previously selected line becomes unselected. Only the selected line can be active (connected to peer, established voice connection, linked to local sound device), the other lines are inactive, on hold, terminated or maybe ringing for an incoming call. When the currently selected line is active and a different line is going to be selected, the previously selected line is automatically put

on hold. When the currently selected line is dialing or alerting and a different line is going to be selected, the previously selected line is automatically disconnected.

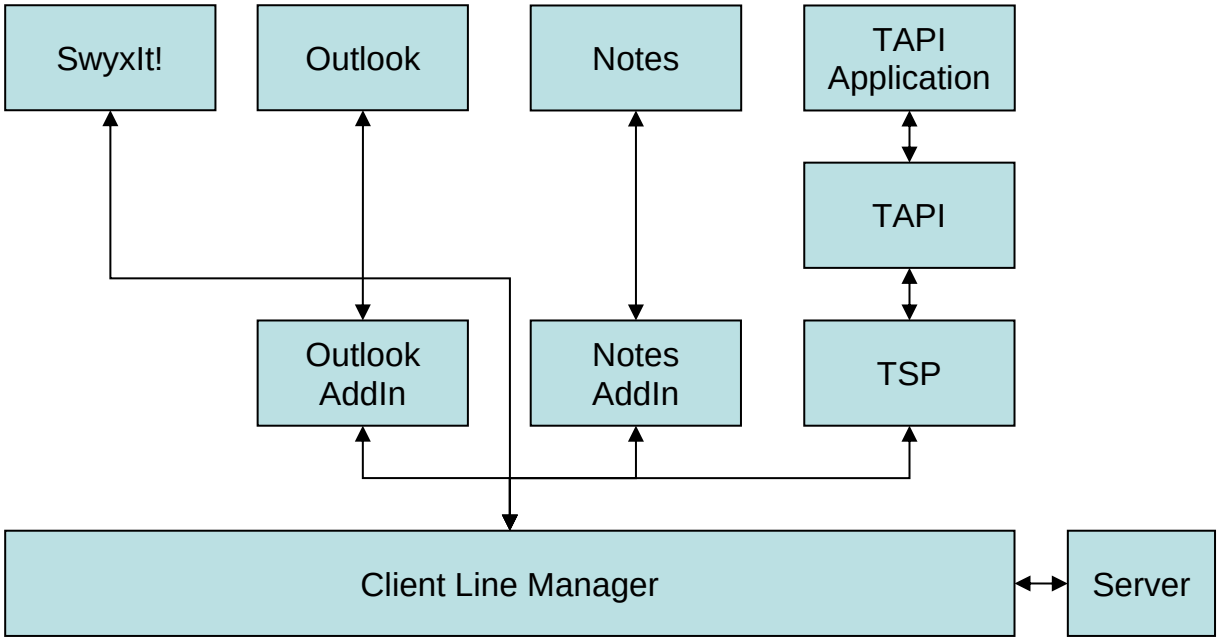
- The power dial mode is used for implementing automatic call distribution systems, predictive dialing, wake up call scenarios and such. Since this mode is intended only for unattended call handling, there is no linkage to the local sound device at all. As a result it is possible to have multiple active lines. The voice connection is handled by the line, by default it sends silence to the peer and incoming voice from the peer is ignored. There is no automatism related to line selection. When a line is hooked off, it does not become selected. When a line becomes unselected, it is not put on hold or disconnected implicitly. As a matter of fact the line selection has no real meaning in this mode. The only effect of line selection is that the SwyxIt! display shows information for the selected line and user input from SwyxIt! is forwarded to the selected line. But since we are talking about an unattended system, this does not matter. In most cases the SwyxIt! user interface is not running anyway in this mode.

The power dial mode is enabled in the registry:

- Key: “HKLM\SOFTWARE\Swyx\Client Line Manager\CurrentVersion\Options”
  - DWORD value “EnablePowerDialMode”: set to 1 for power dial mode, set to 0 for normal operation mode (default when value does not exist).
  - DWORD value “CancelBlindTransferOnVoicemail”: set to 0 when a blind call transfer should not be cancelled when the transfer call is connected to the voicemail or call routing. By default the client will abort a blind call transfer when it gets connected to call routing since usually we want to transfer someone to a real person. For power dial systems it is very common to transfer calls into scripts, so this automatism should be switched off.

1.3 Client Architecture

The core component of SwyxIt! is the Client Line Manager (CLMgr). All applications like SwyxIt!, Swyx Client TSP, Swyx Outlook AddIn and third party applications are running on top of the Client Line Manager. All these applications are using the same Client Line Manager instance. In this way several applications may use the same lines simultaneously. So when a call is established using a TAPI application, the same call will be visible within SwyxIt!. A call can be initiated by the Outlook AddIn and then be put on hold by TAPI.
The CLMgr connects to the server respective SIP proxy, handles calls and manages audio streams. All other components are accessing the server through the CLMgr.



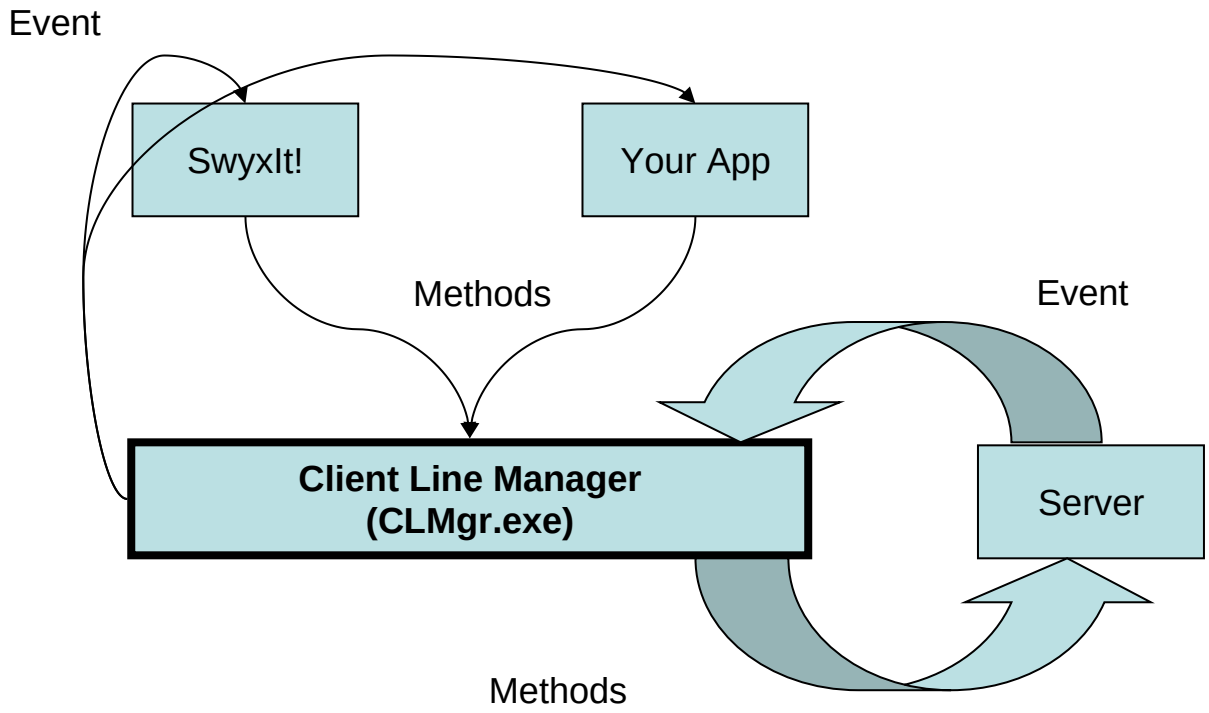
1.4 Methods and Events

The CLMgr provides a set of methods for call control, media streaming and configuration. The API allows tasks like dialing, call hold, call transfer, establishing of a conference… The functions published in the Client SDK allow nearly everything that can be done via SwyxIt! as well. The other way round the CLMgr will send events to the client applications when anything important happens: incoming calls, a call gets connected or disconnected, a speed dial state changes...

Most call related functions are handled asynchronous. This is merely like in old ship movies: The captain tells the mate: “full speed ahead”. The mate tells the cox: “full speed ahead”. The cox tells the engineer: “full speed ahead”. The engineer finally pushes the button and confirms it to the cox and the message queue goes back till the captain.

Example: When an incoming call is accepted in the SwyxIt! user interface, SwyxIt! calls the appropriate CLMgr function. The CLMgr handles that task asynchronously and tells the server to accept the call. When that request is confirmed by the server, the call gets accepted; the CLMgr connects the line, starts the media streaming and tells SwyxIt! that the line state has been changed from “ringing” to “active”. Finally SwyxIt! displays the line in connected state and updates the display.

The important message is: The implementation has to be aware that call related API calls are handled asynchronously. When e.g. an invalid number is dialed, the dial function will return without error but later on the line will get disconnected with an appropriate disconnect reason.

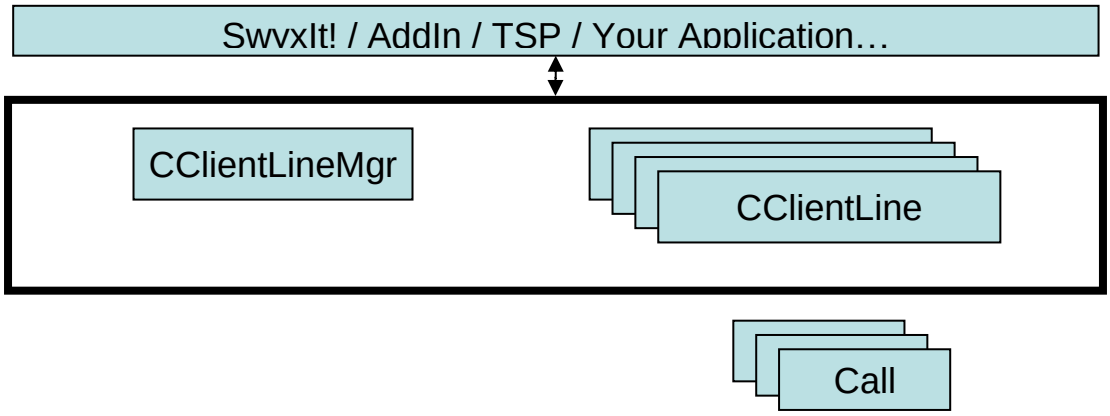


1.5 Lines, Selected Line & Line States

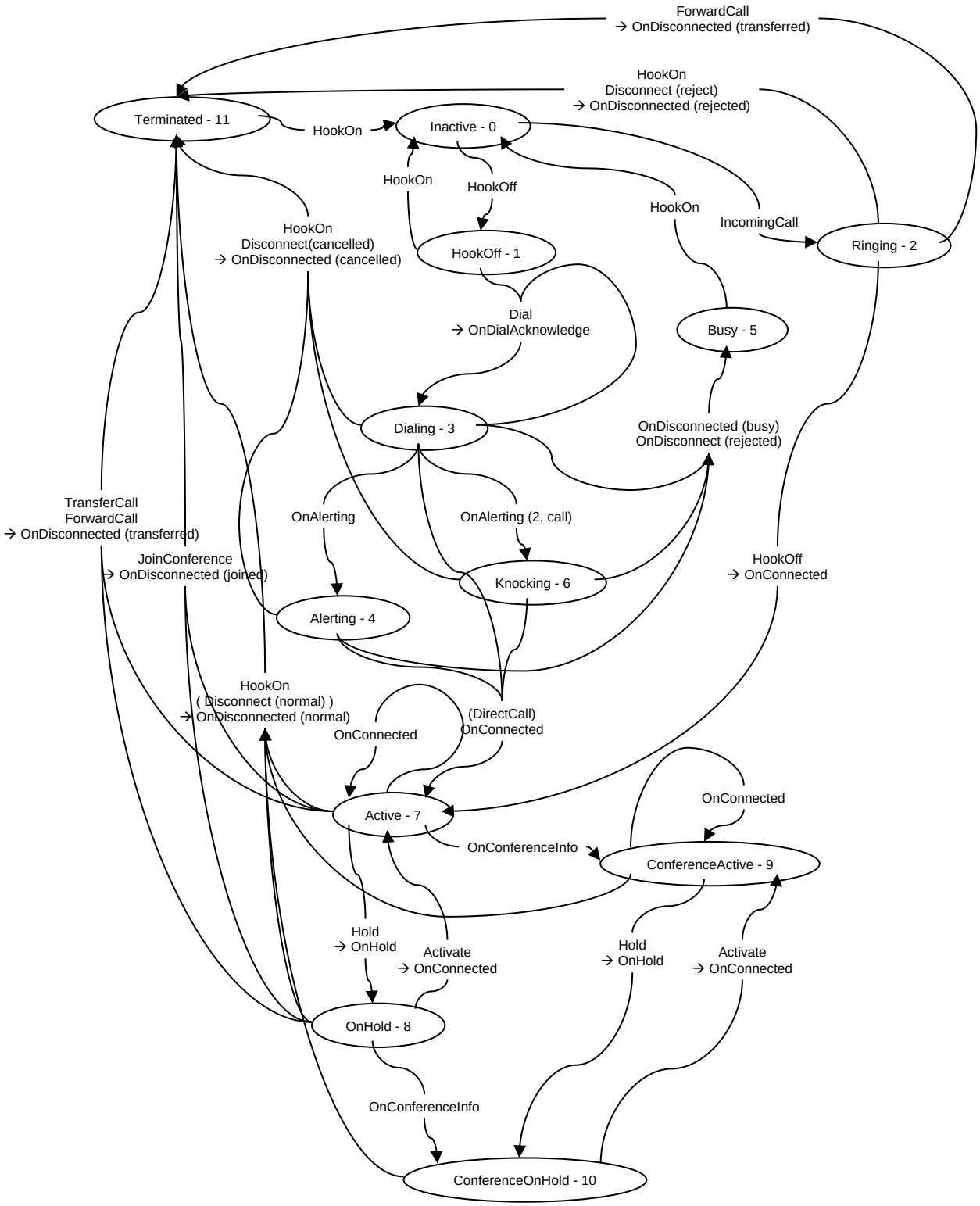
Within the line manager we have one CClientLineMgr object and multiple CClientLine objects. The job of the CClientLineMgr object is reading and writing configuration, registration with the server or SIP registrars, audio device handling, switching between lines… The CClientLine objects are handling the individual calls. There is always one line selected (line in focus). The selected line is linked to the audio device. In normal operation mode it is not possible having two or more active calls. The CClientLineMgr will ensure that an active line is automatically put on hold when a different line becomes active. A line is always in one of the following states, resembling the state of the corresponding call:

- **Inactive**: line is idle, no call
- **HookOffInternal**: outgoing call, hook or line clicked, handset off hook, internal dial tone is played. Waiting for more digits.
- **HookOffExternal**: outgoing call, line is off hook and the public access prefix has been dialed so far. Waiting for more digits.
- **Ringing**: the line is signaling an incoming call
- **Dialing**: outgoing call, one or more digits have been dialed so far. Waiting for more digits or a response from the server.
- **Alerting**: outgoing call, the called party (destination, peer) will hear ringing
- **Knocking**: outgoing call. This is a secondary call for the peer, the peer is already talking on an other line.
- **Busy**: outgoing call, the destination is busy
- **Active**: incoming or outgoing call, logical and physical connection are established
- **OnHold**: incoming or outgoing call, the peer gets music on hold

- **ConferenceActive**: incoming or outgoing call, a conference with multiple participants has been established.
- **ConferenceOnHold**: incoming or outgoing call, a conference with multiple participants has been established but we have currently put the conference on hold. The conference participants will not hear music on hold.
- **Terminated**: incoming or outgoing call, the call has been disconnected actively or passively.
- **Transferring**: incoming or outgoing call, blind call transfer. When a call transfer between two lines is initiated while the destination line is not yet connected, this is a blind call transfer. The transfer source line will stay in state transferring while the transfer is still in progress. The peer on that line will hear an alerting sound.
- **Disabled**: this is a special inactive state. Temporarily no calls are accepted on this line.
- **DirectCall**: incoming intercom call, the connection is established but the micro is muted. The peer can talk to us.



The following diagram displays a “simple” view of the line state machine. The diagram is not complete but should give an idea how a SwyxIt! line works.



1.6 CClientLineMgr Methods

The CClientLineMgr object provides methods for registration, line selection and media streaming.
Please note that the parameter types are depending on the used programming language and COM wrapper technique. All details about the interfaces, structures, enums and Client-LineMgr events can be found inside the attached Client SDK files (idl files, chm file).

2 Programming Guide

2.1 64 bit Applications

SwyxIt! is a 32bit application, but it's COM objects can also be used from 64bit applications. When you install SwyxIt! 12.40 or newer, the setup not only installs a 32bit bit proxy/stub dll (clmgrps.dll), but also a 64bit version (clmgrps64.dll).

This SDK also offers a 32bit and a 64bit runtime callable wrapper (RCW) assembly (Interop.clmgrlib.dll and interop64.clmgrlib.dll) allowing the API to be used from managed code.

2.1.1 Use from 64bit native C++ Applications

The sample application in the VC++/Simple folder of the SDK can either be built as 32bit or as 64bit version and shows how to access the COM object from a native application. It uses the clmgrpub.idl file to generate C/C++ header files you include.

2.1.2 Use from 64bit managed code Applications

Add the Interop64.CLMgr.dll as reference to you managed code project. The SwyxIt! COM API uses custom structs and Microsoft uses different struct member alignments for 64bit and 32bit, so a common RCW is not possible and you either have to use the 32bit or the 64bit version.

2.2 Creating Line Manager Object and Accessing Lines

2.2.1 VB Script

The following code show the instantiation of a line manager object and calling some methods from VB script. When using VB 6 or VB script, you have to use the Dispatch methods (all methods where the name starts with Disp, like DispSimpleDial, DispHookOff, DispHookOn...).

```
Dim PhoneLineMgr : Set PhoneLineMgr = Nothing
Dim PhoneLineFocus : Set PhoneLineFocus = Nothing
```

Create a line manager instance:

```
Set PhoneLineMgr = Wscript.CreateObject("CLMgr.ClientLineMgr")
```

Dial a number:

```
PhoneLineMgr.DispSimpleDial("001191")
```

Get the selected line:

```
Set PhoneLineFocus = PhoneLineMgr.DispSelectedLine
```

Hook on the selected line. This will terminate the call.

```
PhoneLineFocus.DispHookOn()
```

Clean up:

```
Set PhoneLineMgr = Nothing
Set PhoneLineFocus = Nothing
```

2.2.2 VB 6 (no longer tested)

With Visual Basic 6 the Client SDK can be added to a project using menu Project\References. Check the library "CLMgr 2.0 Type Library". Afterwards the Object Browser will list all available objects and methods for library CLMGRLib. Due to technical reasons you will see many more objects and interfaces than you can use. From Visual Basic you can use the objects CClientLineMgr and CClientLine and its methods and properties starting with Disp (like DispHookOff). Only the object CClientLineMgr will be created by your application. CClientLine objects are not created directly, but you will access the line by calling appropriate CClientLineMgr functions. The following example retrieves the selected line and puts it on hold:

```
Public WithEvents PhoneLineMgr As CLMGRLib.ClientLineMgr
Set PhoneLineMgr = CreateObject("CLMgr.ClientLineMgr")
Public PhoneLineFocus As Object
Set PhoneLineFocus = PhoneLineMgr.DispSelectedLine
PhoneLineFocus.DispHold
```

For receiving line manager events with Visual Basic define a CClientLineMgr object:
Public WithEvents PhoneLineMgr As CLMGRLib.ClientLineMgr

Create an object:
Set PhoneLineMgr = CreateObject("CLMgr.ClientLineMgr")
Add the following function to your code that will receive the events. The meaning of parameter param is explained in file “CLMgrPubTypes.h”, included in the SDK.

```
Sub PhoneLineMgr_DispOnLineMgrNotification(ByVal msg As Long,
ByVal param As Long)
 Select Case msg
  Case 0 'PubCLMgrLineStateChangedMessage
  Case 1 'PubCLMgrLineSelectionChangedMessage
  Case 2 'PubCLMgrLineDetailsChangedMessage and so on
End Select
End Sub
```

2.2.3 VB .Net Simple

Please refer to sample “Visual Studio.Net VB Simple”.

Add a reference to Interop.clmgr.dll or Interop64.clmgr.dll which are part of the SDK

Import the library:

```
Imports IpPbx.CLMgrLib;
```

Declare and create a line manager object:

```
Private WithEvents MyClmgr As New ClientLineMgrClass
```

Declare an event handler that receives line manager events from that MyClmgr object. This method will be called when the line manager sends a message to your application:

```
Private Sub clmgr_PubOnLineMgrNotification(ByVal msg As Integer, ByVal
param As Integer) Handles MyClmgr.PubOnLineMgrNotification
```

That method will forward the event to the form by calling Invoke:

```
Me.Invoke(Me.myDelegate, New Object() {msg, param})
```

This results in a thread safe calling of

```
Public Sub MyLineManagerMessageHandler(ByVal msg As Integer, ByVal param As
Integer)
```

This is required for accessing the form from the COM event thread.

The parameter msg will be the line manager event (see enum PubCLMgrMessages in file “CLMgrPubTypes.h”, included in the SDK). The parameter param depends on the message; please refer to the comments within “CLMgrPubTypes.h”.

Get the selected line and do something with it:

```
Private SelectedLine As ClientLine
Dim i As Integer
SelectedLine = MyClmgr.DispSelectedLine
i=SelectedLine.DispState
SelectedLine.DispHookOff()
SelectedLine.DispHookOn()
```

2.2.4 C# .Net Simple

This is similar to VB .Net. Please refer to the provided sample code “Visual Studio.Net 2005 C# Simple”.

Add a reference to Interop.clmgr.dll or Interop64.clmgr.dll which are part of the SDK

Import the library:

```
using IpPbx.CLMgrLib;
```

Declare and create a line manager object:

```
private ClientLineMgrClass MyCLMgr = new ClientLineMgrClass();
```

The sample code contains a class “ClientSdkEventSink”.
Declare an event handler that receives line manager events from that MyClmgr object.

```
private ClientSdkEvents.ClientSdkEventSink MyEventSink = new
ClientSdkEvents.ClientSdkEventSink();
```

In the form load method, link this event handler object to your event handler method of your form:

```
MyEventSink.Connect(pCLMgr, this, new
ClientSdkEvents.LineManagerMessageHandler(OnLineManagerMessage));
```

This method will be called when the line manager sends a message to your application:

```
private void OnLineManagerMessage(ClientSdkEvents.ClientSdkEventArgs e)
```

The parameter e will be the line manager event with the properties e.Msg (see enum PubCLMgrMessages in file “CLMgrPubTypes.h”, included in the SDK) and e.Parameter. “Parameter” depends on the message; please refer to the comments within “CLMgrPubTypes.h”.

Get the selected line and do something with it:

```
private ClientLine SelectedLine;
int i;
SelectedLine = (ClientLine)MyCLMgr.DispSelectedLine;
i=SelectedLine.DispState;
SelectedLine.DispHookOff();
SelectedLine.DispHookOn();
```

Don’t forget to release the event sink object when the form is closed:

```
MyEventSink.Disconnect();
```

2.2.5 C++

For Visual C++ add the files CLMgrPubTypes.h, CLMgrPubTypes.c und CLMgrPub.idl to your project. You will create a Client Line Manager object (CLSID_ClientLineMgr) and use its interfaces like IClientLineMgrPub or IClientLineMgrPub2. The Client Line Manager object provides methods for accessing lines. Using the line interface IClientLinePub you may initiate actions on a line or retrieve line details. This will be shown in the first example "Visual C++ Simple". Please refer to the provided sample code. Please pay attention that you can receive line manager events as either window messages or COM events. Most C++ samples will show both ways. There are different build targets for using window messages or COM events. The related event handling code is enclosed in #ifdef blocks. The advantage of window messages is that you will receive the events within your main message pump. SwyxIt! is using that method as well. Receiving COM events in C++ is a little bit more complicated. The sample code shows how to do this using the Active Template Library, connection points and event sinks.

2.3 Linking Media Streaming of Lines in Power Dial Mode

The following example in VB script shows calling out on two lines and linking the media streaming of the two lines. Then you have to wait till both lines are disconnected later on:

```
Option Explicit

Dim PhoneLineMgr : Set PhoneLineMgr = Nothing
Dim PhoneLine1 : Set PhoneLine1 = Nothing
Dim PhoneLine2 : Set PhoneLine2 = Nothing
```

```
Dim errval, DialDigits, Pos, i
Dim sLink

Set PhoneLineMgr = Wscript.CreateObject("CLMgr.ClientLineMgr") : CheckError
Set PhoneLine1 = PhoneLineMgr.DispGetLine(0)
Set PhoneLine2 = PhoneLineMgr.DispGetLine(1)

sLink=PhoneLineMgr.DispCreateMediastreamingLink()

PhoneLine1.DispSetMediastreamingLink(sLink)
PhoneLine2.DispSetMediastreamingLink(sLink)

PhoneLine1.DispHookOff()
PhoneLine1.DispDial("12345")
PhoneLine2.DispHookOff()
PhoneLine2.DispDial("67890")

Wscript.Sleep 1000

While( (PhoneLine1.DispState>0) And (PhoneLine2.DispState>0) )
Wscript.Sleep 1000
Wend

PhoneLine1.DispSetMediastreamingLink("")
PhoneLine2.DispSetMediastreamingLink("")
PhoneLineMgr.DispDeleteMediastreamingLink(sLink)


Set PhoneLineMgr = Nothing
Set PhoneLine1 = Nothing
Set PhoneLine2 = Nothing

Sub CheckError
Dim message, errRec
If Err = 0 Then Exit Sub
message = Err.Source & " " & Hex(Err) & ": " & Err.Description
Fail message
End Sub

Sub Fail(message)
Wscript.Echo message
Wscript.Quit 2
End Sub
```

So when you are using power dial mode, the lines are fully operational, only that the media streaming is not linked at all to the local sound device. But when you connect the media streaming of two lines (using SwyxIt! 6.02 or later), the line manager will link the media streaming of the two participants. When line 1 gets early tones, the peer on line 2 will hear it. When line 2 is connected to a script and gets any music on hold and line 1 is connected to the external party, the external party will hear the music on hold.
The media streaming is then routed through the line manager.
The drawback is: you need to keep the two lines active on the power dial client. You could initiate a transfer later on (when you are sure that the external party is connected to the agent) but this would result in a short interruption of the call. So best practice would be to wait till both lines are disconnected (as shown in my sample code).
My favorite view of this: It is like "Fräulein vom Amt" sitting in front of a huge switchboard. You can have multiple calls on the lines and patch the connections independently from the call states.

There is no build in maximum limit of lines in power dial mode. So you can call DispSetNumberOfLines(500) and have 500 lines. But have in mind that you then might have 500 active connections with 250 media streaming links. You have to test what is possible with a given hardware.

You can link the media streaming of more than two lines as well. For more information please refer to sample “Visual Studio.Net 2005 C# PowerDialTest”.

2.4 Sample Code

The SDK contains several sample projects. You will need Visual Studio for opening the project files.

2.4.1 Visual Basic Script

This sample just shows how to dial a phone number via script.

2.4.2 Visual C++ ATL PlugIn

This sample explains the implementation of a Line Manager PlugIn providing name resolution for SwyxIt! The code is based on the Active Template Library.
It implements a Client Line Manager PlugIn that provides name resolution for SwyxIt!. For unknown peer numbers the Client Line Manager will ask all installed PlugIns for a matching name resolution. So you will be able to provide name resolution based on your own database application. This client too will not register with the SwyxServer.
For a customization one has to replace the GUIDs and friendly class names PlugInSample.MyResolver etc. in the files PlugInSample.idl and MyResolver.rgs with new GUIDs and names in order to avoid conflicts with other 3rd party applications based on the Client SDK. But never ever change any GUID from the files CLMgrPub.idl and CLMgrPubTypes.c!
In file PlugInSample.cpp the functions for registration and deregistration of the PlugIn are implemented. In order to be loaded by the Client Line Manager, the PlugIn writes its class ID into the Line Managers registry HKLM\SOFTWARE\Swyx\Client Line Manager\CurrentVersion\Options\PlugIns\{GUID}
The actual PlugIn object CMyResolver is defined and declared in the files MyResolver.cpp and MyResolver.h. The PlugIn object implements the interfaces IClientAddInLoader, IClientResolverAddIn and IClientLineMgrEventsDisp. The interface IClientAddInLoader will be used by the Line Manager on loading and releasing the PlugIn, the interface IClientResolverAddIn provides name resolution. The interface IClientLineMgrEventsDisp finally receives the Line Manager events.
When loading the PlugIn, the Line Manager calls the functions IClientAddInLoader::Initialize, IClientAddInLoader::GetName and IClientAddInLoader::GetVersion. Before releasing the interface IClientAddInLoader and unloading the PlugIn, the Line Manager calls the method IClientAddInLoader::UnInitialize. The PlugIn has then a chance for cleaning up memory and other resources.
In the example implementation of CMyResolver::Initialize the PlugIn registers an event sink towards the Line Manager for receiving events. The Line Manager interface pointer pIClientLineMgrPub will be stored in the global interface table. Using its cookie m_dwCLMgrCookie the pointer will be retrieved on demand from the global interface table and automatically

marshalled into the current thread context. You will find the used helper functions in the files githelp.cpp and githelp.h. This effort is required as soon as a COM interface pointer is used from multiple threads. GetName are GetVersion unspectacular. In the implementation of CMyResolver::UnInitialize the event sink will be released, that's always a good idea.
Line Manager events drop in in function CMyResolver::DispOnLineMgrNotification. Your PlugIn could e.g. write journal entries into a database or trigger a database application for popping up contact information.
In CMyResolver::GetPreferredNumberStyle the PlugIn can ask for a phone number format to be used for name resolution. Usually one would use the format PubCLMgrNumberStyleFull (like 0004923147770 or 0023147770). For any number to be resolved the Client Line Manager will call CMyResolver::ResolveNumber.

2.4.3 Visual C++ Call Log

This is a simple MFC application that displays a simple call log. A checkbox allows to accept all incoming calls automatically. This client runs besides SwyxIt! and does not log on nor off. Here too there are two build targets "Win32 Debug Using Event Sink" and "Win32 Release Using Event Sink" showing both usage of connection points and COM events. The class CCLMgrEventSink may be used modified in your project.

2.4.4 Visual C++ PlayToRtp

Dialog based MFC Application, allowing the recording and playback of voice stream. As “Play File” please choose a wave file in format PCM, 8000 Hz, 16 bit, mono. The playback is started by pressing “Start”. The file can be played in a loop, the pause between loops can be configured. The playback is heard only by the peer.
This sample does not work in power dial mode.

2.4.5 Visual C++ Simple

Dialog based MFC Application that displays information about the selected line and allows simple call control. The sample includes log on and log off code, so this sample must not run along with SwyxIt! but instead of SwyxIt!. Logon and logoff should only be used via SDK when your application intends to run stand alone (without SwyxIt! running aside).

2.4.6 Visual C++ WakeUp

Example for establishing bulk calls using SwyxIt! in power dial mode.

The client has to be switched to power dial mode via registry:

- No implicit line selection
- If a line is selected explicitly, the previous selected line does not become disconnected on hold
- The mediastreaming is not linked to the local sound device
- No local, line related sounds are played (ringing, dialing, alerting, …)
- Hook off/on per local handset will be ignored

Enable the power dial mode using reg file "PowerDialerMode.reg". The reg file will set as well the key "CancelBlindTransferOnVoicemail" to 0 for allowing blind call transfer (forward) to a voicemail announcement. Within SwyxIt! one should disable pop up during connection for avoiding annoying pop ups.

The sample allows dialing on all available lines simultaneously. The phone call jobs are read from file JobsToDo.txt. Every line corresponds to a call to do, described by three strings, separated by ';': The first string is the number to be dialed. On connect the call will be transferred to the number given in the second string. The third string is a counter for the retries. When clicking on start in the sample application, the dialing will be initiated after a short timeout. Clicking on stop will abort the process. In a second file all successful calls will be listed. The file JobsToDo.txt lists all not yet successful call and the current number of retries. When trying this sample on a productive server, please be aware that it is a really good stress test tool. For real scenarios one should add some Sleep() calls in order to give the server time.

2.4.7 Visual Studio.Net C# PowerDialTest

This sample is intended to run on SwyxIt! in power dial mode. Please note that it sets the line count to 50. The actual line count can be higher than the number of lines visible on the skin.
Please note the class “ClientSdkEventSink” and how it is used to receive messages from the line manager thread safe.
The project was used for testing the speed of switching on media streaming between two clients. So it demonstrates how to handle media streaming in power dial mode. In fact the real test scenario was: Run SwyxIt! Now in power dial mode. Configure two SIP accounts for your local SwyxServer. Dial out on behalf of account 1 to the number / URI of account 2. So our call will come in on a second line of the same client. When you then pickup the call, you can measure how long it takes to establish the voice connection by sending out a test sound on one line and record it on the other line. Please don’t use this code to claim support incidents against Swyx ;-)

Outgoing Call: Please enter the dial number. In case that you are using SwyxIt! Now in power dial mode, you have to enter the caller id as well. The “caller id” is then used to select the SIP account that is used for placing the outgoing call. If checkbox “Play Sound” is checked, the sample will play a test sound CallingPartyTestsignal1000Hz10Seconds.wav toward the destination as soon as the line becomes active. This resembles a user that immediately starts speaking when the line becomes visibly active. The incoming media streaming for this call will be recorded. For this you can see in the sample code that on line a media streaming link is created and DispRecordSoundFile is called on the line.
Press “Start call” for placing the outbound call.

If the SIP accounts are configured correctly (as discussed: Use SwyxIt! Now in power dial mode, two SIP accounts), the call will come in on a second line. Accept the call by pressing “Accept Call”. The sample will then display the time difference between accepting the incoming call and receiving the “connect” message on the outgoing call on the first line. This is the time difference for establishing a logical connection. If “Play Sound” is checked, the application will play a test sound CalledPartyTestsignal2000Hz10Seconds.wav towards the caller. This resembles a user that immediately starts speaking after accepting a call. The incoming media streaming for this call will be recorded.

Independent from the just described test scenario, the other buttons show how to link the media streaming of multiple lines to each other, playback sound files in power dial mode or record voice in power dial mode.

When looking into the code itself you will find as well an example of handling incoming DTMF.

2.4.8 Visual Studio.Net C# Simple

Simple application in C# showing dialing and receiving events. Nevertheless pay attention to the code comments for avoiding nasty problems in your own code. Please note the class “ClientSdkEventSink” and how it is used to receive messages from the line manager thread safe.

2.4.9 Visual Studio.Net VB Simple

Very simple application in VB showing dialing and receiving events. Nevertheless pay attention to the code comments for avoiding nasty problems in your own code. Please note the thread safe event handling in method “clmgr_PubOnLineMgrNotification”.

2.4.10 Visual Studio.Net C# IpPbxMPC

This example is a complete C# application named IpPBX Media Player Controller. It can be used to control a media player such as Winamp or Windows Media Player. Whenever not all SwyxIt! lines are inactive IpPBXMPC sends the configured player a "pause" command. When all SwyxIt! lines become inactive again, the player gets an "unpause" command from IpPbxMPC. Alternatively the active state of one or more speed dial keys can be used, too.

2.4.11 Visual Studio.Net C# Plugin

This example demonstrates how to implement a line manager PlugIn using C#. For more information please refer to the public Swyx Forum Blog:
http://www.swyx-forum.com/community/Blogs/tabid/55/EntryId/438/SwyxIt-Full-Text-Search-Plug-in-in-C.aspx

3 Reference

3.1 Client SDK Methods

The reference for all supported interfaces and its methods is given in the included file "CLMgrPub.idl". The methods and its parameters are documented within that file. Even if you are programming in VB or C# you should read this file as a reference for understanding the parameters. Alternatively you can take a look into the attached .chm helpfile, which is strongly recommended, because of its easy to use index, searchoption and hyperlink functionality.

The file "CLMgrPubTypes.h" contains the enumerations for all supported events, line states, and disconnect reasons.

The file "CLMgrPubTypes.c" contains defines for some Class IDs that are required for linking C++ projects.

3.2 Supported TAPI Features

The Microsoft TAPI provides an upper level interface to be used by TAPI client applications like Windows Dialer or CRM software. Telephony applications like SwyxIt! integrate into TAPI by implementing a so called TAPI Service Provider (TSP). A TSP is a sort of driver that forwards TAPI function calls to the 3rd party telephony hardware or software components. The SwyxIt! TSP supports TAPI 2.x. Nevertheless it can be used by TAPI 3.x client applications as long as no functions are used that are not supported by TAPI 2.x. SwyxIt! supports basic call handling including transfer and conference, there is no access to media streaming.

Please note that the "SwyxIt! TAPI Service Provider" provides some configuration that is accessible using the telephony control panel applet. Especially you can configure…

- A fixed dial prefix for outgoing calls (if you need this for any TAPI application that does not dial canonical and does not use the Windows locations neither)
- Caller ID format: the format that is used for reporting phone numbers towards TAPI. Please use the format that is appropriate for your application. For DATEV e.g. you would choose plain or full. For ACT! you would choose plain.
- "Signal outbound calls": By default the TSP does only signal incoming calls to the TAPI. So the TAPI would not be aware of outbound calls that are started using SwyxIt!. If you need the visibility of all outbound calls to your TAPI application, please check this checkbox.

SwyxIt! supports the following TAPI functions:

- lineOpen
- lineClose
- lineMakeCall
- lineDial
- lineAnswer
- lineHold

- lineUnhold
- lineSwapHold
- lineSetupTransfer
- lineBlindTransfer
- lineCompleteTransfer
- lineSetupConference
- linePrepareAddToConference
- lineAddToConference
- lineRemoveFromConference
- lineDrop
- lineCloseCall
- lineGetAddressCaps
- lineGetAddressStatus
- lineGetCallInfo
- lineGetCallStatus
- lineGetDevCaps
- lineGetID
- lineGetLineDevStatus