NOVELL® RESEARCH

NetWare Telephony Services for Application Developers

Kevin White
Software Engineer
Novell Developer Support

This article offers an in-depth overview of the tools and techniques that developers will encounter in their application development with Novell's Telephony Services Application Programming Interface (TSAPI). This DevNote covers such topics as the following: the Telephony Server (TServer) NLM, which receives and processes client TSAPI requests; TSAdmin, the Telephony Services administration tool; the Simulator, which allows developers to test and debug programs without running in a production environment; and such TSAPI programming elements as ACS Streams.

Novell, Inc.
122 East 1700 South
Provo, Utah 84606 USA

Contents

Introduction

Since Novell introduced its NetWare Telephony Services SDK several weeks ago, it has been exciting to see the potential of this product start showing up in various applications. The SDK provides services that have not been easily accessible until now. In conjunction with Novell Developer Support's desire to help developers in every way possible, there will be more articles and information coming out about the Telephony Services API (TSAPI), which we hope will make it easier for developers to integrate TSAPI into their applications.

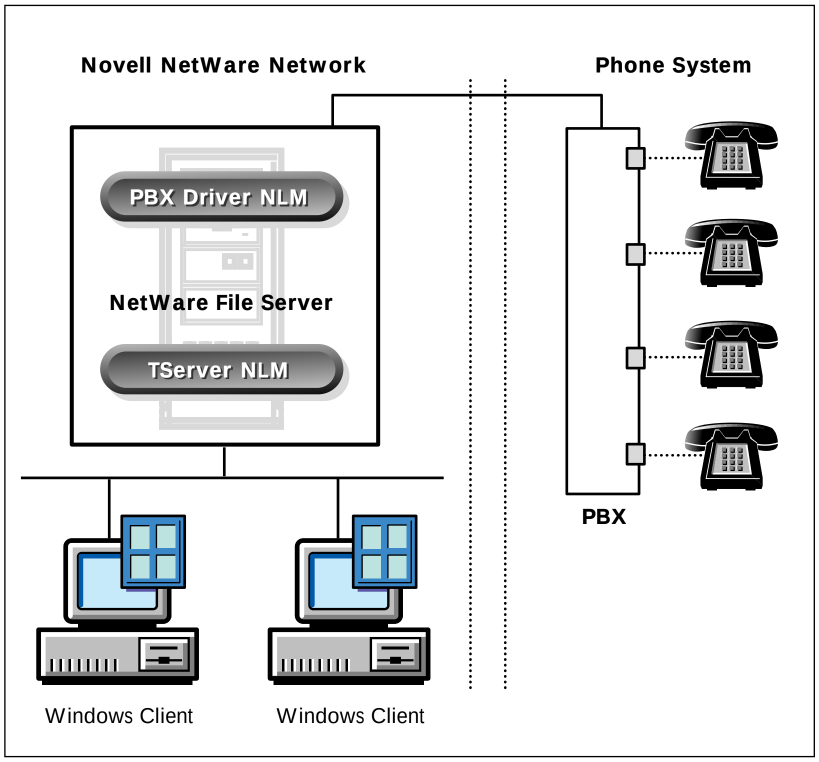
This article offers an in-depth overview of the tools and techniques that developers will encounter in their application development with Novell's TSAPI. Future articles will focus on just one or two of the issues mentioned here and take them to a much lower level.

The first consideration in developing a Telephony application is understanding the architecture.

Telephony Architecture

As Figure 1 shows, the Telephony services are actually comprised of several components.

Figure 1: Telephony services are made up of such individual elements as the server, the PBX, and associated NLMs and drivers.



On the Novell NetWare network side of the figure, there is a NetWare file server with its accompanying client workstations. On the other side, there is a Private Branch Exchange (PBX) with several phones attached to it. The interface is provided by some sort of a hardware link and a driver for that hardware on the file server.

The type of hardware link depends on the PBX in use. Whatever the link, it is the basis for the communication necessary for NetWare to provide its TSAPI services.

The NetWare server also has a Telephony Server (TServer) NetWare Loadable Module (NLM) loaded. The TServer's responsibility is to receive TSAPI requests from client workstations, convert them into a predefined interface, and pass them along to whatever PBX driver happens to be loaded. The PBX driver does the "dirty work" of controlling the PBX to provide the services that have been requested.

Client workstations on the Novell NetWare network send requests to TSAPI, which are transported over the wire to the TServer. These requests on the client are made from an application to some sort of library that will provide the necessary code to transfer the request to the TServer. At this time, the requests are provided to Windows applications through DLL files and to NetWare NLMs through TSLIB.NLM. Other platforms will be supported in future releases of NetWare Telephony Services.

Development Tools

Several tools are available for use with TSAPI application development. These include trace utilities, the PBX Simulator, and a test application. They are described in Table 1.

TSAPI Development Tools

Tool	For use on	Description
TSCALL.EXE	Windows Client	TSCALL is a Windows application that uses Telephony Services to provide basic call control, such as making and clearing calls. If other telephony services don't appear to be working, try TSCALL and see if it works. This can help determine if the problem lies in the application or in the hardware or software setup of NetWare Telephony Services.
CSTASPY.EXE	Windows Client	CSTASPY is a Windows application that can be used to trace messages going back and forth from TSAPI applications to the TServer. If an application isn't functioning as desired, try CSTASPY to give you a message trace. Check to make sure that all messages are being sent and received as they should be.

CLSIM.EXE	Windows Client	CLSIM is used in conjunction with the SIM.NLM on the TServer. This is the Windows client piece, which will allow you to test sending certain messages to the Simulator module to determine that they are working. This can also be used in the background, while your application is running in the foreground, to give a similar message trace as CSTASPY does on a "live" PBX.
SIM.NLM	Telephony Server	SIM.NLM is a "PBX Simulator." This is a very useful developmental tool because you don't have to have a real PBX to develop and test a TSAPI application. The Simulator emulates a PBX and makes the TServer and TSAPI applications believe they are actually controlling a real PBX. This is useful when you don't want to put a test application in a production environment.

These tools can help developers rapidly produce quality applications.

Software Installation

Installing the NetWare Telephony Services is a fairly simple process, which is described in the documentation. However, there were some last minute changes that are included in the readme file, which should be printed and kept with the printed manuals.

If followed, the readme file and printed manuals make installing and loading the basic Telephony Services very easy. The documentation for the Simulator, however, is not complete at this time. There is a readme file on the Simulator diskette, and the necessary information will appear in the printed manuals in the future.

The Simulator requires several files to function properly. The following table lists each of the files and where they should be copied to.

Files for use with the Simulator

File Name	Located on	Copy to
SIM.NLM	Simulator Diskette	SYS:\SYSTEM on the TServer
CLSIM.EXE	Simulator Diskette	Any drive accessible to the Windows client
CLSIM.HLP	Simulator Diskette	Any drive accessible to the Windows client
ATTPRIV.DLL	G3PDDSK2 Diskette	Client's \WINDOWS directory

In order to write applications that use TSAPI, the header and library files must be available to be compiled and linked with your application. These files are on the TSAPI SDK diskette. As with any external libraries, these files need to be copied to some directory that is accessible to your compiler.

Administration

After installing the software, you need to complete certain administrative tasks before using Telephony Services. For example, you need to specify which users are to be allowed to use Telephony, and which devices they can control. You do all of this through the TSAdmin (Telephony Server Administration) tool, which can be installed when you install Telephony Services. Instructions for the use of this tool are included in the manuals.

Simulator

The Simulator has a specific purpose--helping application developers test and debug programs without running in a production environment. Maybe you don't want your program to run in the production environment because it is unstable and needs to be fully tested before people start using it. Maybe you just don't have a physical PBX for use in your system. Or maybe you just want each engineer to have his or her own environment to program and test in without affecting each other, and you don't have the money or office space to provide each engineer with a PBX (who does have that kind of money?).

Whatever the reason, there are many situations when it would be desirable to make TServer and TSAPI "think" they are in a PBX environment without having a physical PBX present. This is what the Simulator is used for.

At this point, it is assumed that the Simulator is properly installed. The correct files should be copied both to the client and server machines, in the directories specified in the table above. It is also assumed that the TServices Admin utility has been used to configure users and their devices for use with the TServer.

You have to configure the Simulator before using it. Because its function is to emulate a PBX, you must configure it just as you would configure a physical PBX. Phone extensions need to be set up, as well as ACD and Trunk groups. It is easy to confuse this configuration with the user security database configuration performed through the TServices Admin utility. Remember that this is very different--the Simulator has to be configured like a PBX.

Use the CLSIM utility (briefly described above) to configure the Simulator. CLSIM is a Windows application used for just about everything associated with the Simulator. The first time you run the CLSIM, use the Configuration options to set up the "fake PBX." After configuring the Simulator, save the configuration file to the SYS:\SYSTEM\TSRV\ directory with a .SIM extension so that it is available to the SIM.NLM when it is loaded.

Simulator Configuration Process

After entering the CLSIM utility on the Windows client workstation, use the First Digit Table option from the Configuration menu. The "first digit table" is where you say, "Extensions that start with a 9 and have 3 digits are of this type." This is how a physical PBX works in its most simple form.

This is also how the Simulator knows how to emulate the functionality of a real PBX.

You need to configure the same setup here as you do in the TServices Admin utility. For example, if I created a user in the user security database that has a phone extension of 3457, I would need to use the First Digit Table to set up First Digit of 3 as "extension" with the Number of Digits set to 4. This way, when the Simulator uses this configuration file, it handles any devices that start with a 3 as phone extensions.

You can also configure act groups or trunk access codes here. After you have configured the First Digit Table to make available any type of combination you will use in the TServices Admin utility, you can select the OK button to close this window.

Next, you need to create the actual Stations, Trunk Groups, and ACD Groups. You can create these using the appropriate options from the Configuration Menu. Remember to create the appropriate stations and other devices according to the same numbers you used in the TServices Admin utility (or you will get an error message when you try to "dial" one of the extensions).

After you have configured all of the appropriate PBX information, you should save this file, using the File/Save option, to the SYS:\SYSTEM\TSRV\ directory, with a filename that has a .SIM extension. This file is now available to the SIM.NLM when it loads on the file server. You can have more than one of these configuration files on the server, but the SIM.NLM can only use one of them at a time. This only makes sense, because the Simulator can only emulate one physical PBX at a time.

When the configuration file has been created and saved correctly, the SIM.NLM module must be loaded on the server. To do this, go to the NetWare server and at the system console prompt type LOAD SIM *filename* (replacing *filename* with the configuration filename used earlier).

For example, if I created a TEST.SIM file through the CLSIM.EXE utility on the Windows client, I would type LOAD SIM TEST.SIM. If the TEST.SIM file was not saved to the SYS:\SYTEM\TSRV directory, it will not be loaded properly.

Once the Simulator has been loaded with the correct .SIM file, you can go back to the Windows workstation and begin using the Simulator. Instructions for its use are in the printed manuals. First, you have to "start" a Simulator session from the RUN menu in CLSIM. After correctly entering the NetWare TServer's server name, you will see a CONFIRM Open Stream message in the client portion of the CLSIM. You can now begin sending messages to the TServer, through the Send Message menu.

For example, if I had defined phone extensions 7559 and 3800, both in the configuration file, and in the TServices Admin utility, I could choose Make Call from the Send Message menu, and have 7559 call 3800. The Simulator would then tell the simulated phone 7559 to call the simulated phone 3800. Of course, you won't hear any physical devices start ringing, but you will have messages appearing telling you that the call was created. You use the rest of the Send Message options in the same way. They are documented in the Simulator User's Guide.

I suggest experimenting with the Simulator, both with the CLSIM program as well as the menu options available on the TServer in the Simulator. These tools provide a powerful set of resources to developers who for one reason or another aren't using a physical PBX.

Programming for TSAPI

There are several key concepts to programming for TSAPI that you should be familiar with. The SDK is different from most SDKs, but is very powerful and flexible. It accommodates many programming styles and needs. The more familiar you are with the overall functionality and services provided by the TSAPI SDK, the easier and more enjoyable time you will have integrating phone control into your applications. You should take the time to read through the documentation, because it will pay off both in saved time and avoided frustrations.

One of the most important concepts to grasp is where the responsibility lies when calling TSAPI services. Refer to the diagram presented in the beginning of this article. Now let's think of a programming example. Your killer Windows super-program makes a function call to the TServer. The TServer needs to give your function a return code--either it worked or it didn't. So the TServer checks to see if it can handle the function call you requested.

Are all the prerequisites filled? Are there enough system resources to do what you wanted to do? Is anything obviously wrong or preventing the successful completion of the function call? No? Good! Send back a successful return code. But what's wrong with this picture? Just because the TServer successfully accepts what your function call requests doesn't mean that the PBX driver will, or that the PBX itself will.

When you get a return value to a function call with TSAPI, remember that this is just the return code from the TServer. There are still other ways to make sure the PBX driver was successful in your request. This is done through Confirmation Events.

There are different types of events. We will only discuss the simplest, Confirmation Events. Whenever the PBX driver needs to pass information along to the Client (via the TServer), it sends an event notice. Your application can do whatever it wants when it receives the event notices. There are even different ways to receive these notices (this will be discussed in a future article).

The important question here is "What type of event was it?" Confirmation events are the events that tell us that the service we requested actually did get performed as requested. Now our application knows it was successful. The important thing to remember is that you need both a successful return code from the function call to the TServer as well as a Confirmation Event from the PBX driver in order to be sure that the function call was 100 percent successful.

One of the most basic concepts to understand once you start TSAPI programming is ACS Streams. A Stream in TSAPI programming has a different meaning from other streams you may have worked with (or waded through) in the past. I like to compare a stream to a door.

You can’t walk through a closed door just as you can’t make TSAPI calls without first opening a stream. You open a stream with the acsOpenStream() function call. But just because this function call doesn’t return an error message doesn’t mean it worked. Again, it’s like opening a door. Just because you twist the door knob doesn’t mean the door will open. It may be locked or there may be some object in the way.

The same holds true for opening a stream. You can make the call successfully, but it still may not be able to open the stream. Maybe there aren’t enough resources. For whatever reason, you need to wait until you receive the Confirmation Event, as well as the successful return code to your function, before making further TSAPI requests.

When you open a stream, among many other parameters, you need to tell TSAPI which TServer that you want to connect to, as well as a username and password. Telephony Services keeps a user security database that keeps track of users and which devices they can control. It also uses the NetWare bindery.

The bindery is the database kept by the NetWare server of all objects on the server, such as users, groups, printers, etc. When Telephony opens a stream, the TServer checks the username and password to verify they are a valid user on the NetWare file server. Once this is verified, the TServer checks the user security database to see if the username appears there. If it does, the stream can be opened.

Any function calls that are subsequently made with this stream will have access to any devices (and only those devices) that the user is set up to control through this security database (which was configured through the TServices Admin utility).

Once your stream is successfully opened, the stream handle is passed as a parameter in nearly every function call to TSAPI. The stream is how the TServer keeps track of all the necessary information that as a programmer you don't have to worry about, but is essential to the TServer in order to track your use of Telephony Services. The stream is your logical link to the TServer, and it is your only logical link. That's why it's so important.

You can now make other function calls for call control or other services, as provided by TSAPI. This brings up another important point. You need to know the difference of functionality from a programmer's point of view between the TServer and the PBX driver.

The TServer includes a full level of functionality, as defined by the NetWare Telephony Services SDK. However, different PBXes have varying levels of functionality, and their drivers may not implement all of the functionality the PBX provides. It is not common for a PBX driver to provide support for 100 percent of the services to an application that the TServer does.

The various PBXes each provide a different subset of the services that the TServer does, though each of them will provide some of the most basic services, such as making and clearing calls. Function calls are available to help determine the functionality provided by a particular PBX driver, which your application can make in order to determine if it will be "safe" to make certain function calls to TServer. If you want your application to be very generic and portable between PBXes, the application will have to make use of only the most basic services.

Some of the advanced features of different PBXes can be implemented through software rather than relying on PBX capabilities. However, this will be less efficient, take longer to develop and test, and some of the features can be difficult at best to implement in such a way. This is an important consideration to take into account when designing your Telephony application.

After your application has made all of the TSAPI calls it needs, and is ready to quit, don't forget to "close the door." The stream that was previously opened needs to be closed, not a difficult process but an important one.

Sample Code

The following program is a simple one that opens a stream, makes a call, and closes the stream. It first asks for the username and password, which are necessary for opening the stream. Then it asks for the extension to call from and the extension to call.

It is written for Borland C/C++ 4.0, using EasyWin libraries that take care of opening and creating a window, and emulate a DOS box by supporting the standard printf(), scanf(), etc. teletype functions. This makes the program shorter and easier to understand. Were I to write a professional application, of course, I would use a standard Windows interface of menus, buttons, etc.

You can use this code as an example or as a shell from which to create your own test or production programs. Simply add more function calls to this same program, if you wish, to test further services.

```
/****************************************************************************
**File:MAKECALL.C
**
**Desc:Sample telephony program
**
**This NetWare Telephony Services program opens a stream,
**makes a call, and closes the stream.
**
**This program is written for Borland C 4.0 for Windows 3.1. It
**is compiled as an EASYWIN program, so that I didn't have to
**include all the Windows "overhead" code but can just get to
**the good stuff.
**
**
**DISCLAIMER
**
**Novell, Inc. makes no representations or warranties with respect to
**any NetWare software, and specifically disclaims any express or
**implied warranties of merchantability, title, or fitness for a
**particular purpose.
**
**Distribution of any NetWare software is forbidden without the
**express written consent of Novell, Inc. Further, Novell reserves
**the right to discontinue distribution of any NetWare software.
**
**Novell is not responsible for lost profits or revenue, loss of use
**of the software, loss of data, costs of re-creating lost data, the
**cost of any substitute equipment or program, or claims by any party
**other than you. Novell strongly recommends a backup be made before
**any software is installed. Technical support for this software
**may be provided at the discretion of Novell.
**
**Programmers:
**
**IniWhoFirm
```

```
**--------------------------------------------------------------------------
**KVWKevin V WhiteNovell Developer Support.
**
**History:
**
**WhenWhoWhat
**--------------------------------------------------------------------------
**07-21-94kvwFirst code.
**07-22-94kvwAdded disclaimer
*/

/****************************************************************************
**Include headers, macros, function prototypes, etc.
*/

/*---------------------------------------------------------------------------
**ANSI
*/
#include <stdio.h>
#include <string.h>

/*---------------------------------------------------------------------------
**Windows
*/
#include <windows.h>

/*---------------------------------------------------------------------------
**Telephony
*/
#include <acs.h>
#include <csta.h>

/****************************************************************************
**This function is the entire program, as it is very simple.
**Prompt user for input, open a stream, make a call, close the stream.
*/
void main(void)
{

/*---------------------------------------------------------------------
** The following are defined for the first call, acsOpenStream
*/
ACSHandle_t acsHandle;
InvokeIDType_tinvokeIDType;
InvokeID_tinvokeID;
StreamType_tstreamType;
ServerID_tserverID;
LoginID_tloginID;
Passwd_tpasswd;
AppName_tapplicationName;
Level_tacsLevelReq;
Version_tapiVer;
unsigned shortsendQSize;
unsigned shortsendExtraBufs;
unsigned shortrecvQSize;
unsigned shortrecvExtraBufs;
PrivateData_t*privateData;
RetCode_trCode;

/*---------------------------------------------------------------------
** The following are additional variables necessary for
** acsGetEventBlock
*/
CSTAEvent_teventBuffer;
unsigned shorteventBufferSize;
unsigned shortnumEvents;

/*---------------------------------------------------------------------
** The following are for the cstaMakeCall
*/
DeviceID_tcaller,callee;
```

```
/*-----------------------------------------------------------------
** Miscellaneous variables
*/
short done=0;


/*-----------------------------------------------------------------
** Setup parameters for call to open stream. Also, prompt user for
** input.
*/
invokeIDType=LIB_GEN_ID;
invokeID=0;  /* don't need it, but set to zero anyway */
streamType=ST_CSTA; /* want to use csta functions */

strcpy(serverID,"ATT#G3_SWITCH#CSTA#PRV-NMS");

printf("\nEnter your user name:");
scanf("%s",loginID);
printf("\nEnter your password:");
scanf("%s",passwd);
printf("\nEnter your extension:");
scanf("%s",caller);
printf("\nEnter the extension you want to call:");
scanf("%s",callee);

strcpy(applicationName,"Simple App");
acsLevelReq=ACS_LEVEL1;
strcpy(apiVer,CSTA_API_VERSION);
sendQSize=0; /* default size queue*/
sendExtraBufs=0; /* use default number */
recvQSize=0; /* use default size */
recvExtraBufs=0; /* use default number */
privateData=NULL; /* no private data */

/*-----------------------------------------------------------------
** Open the stream with above parameters
*/
rCode=acsOpenStream(&acsHandle,invokeIDType,invokeID,streamType,
serverID,loginID,passwd,applicationName,acsLevelReq,
apiVer,sendQSize,sendExtraBufs,recvQSize,recvExtraBufs,
privateData);

if (rCode < 0)
{
printf("acsOpenStream failure...");
return;
}
else
{
printf("acsOpenStream success\n");
invokeID=rCode;
}

/*-----------------------------------------------------------------
** Block until the confirmation has been received that the stream
** was successfully opened. Just because NetWare returned the function
** code doesn't mean the stream has been opened yet.
*/
eventBufferSize=sizeof(CSTAEvent_t);
rCode=acsGetEventBlock(acsHandle,&eventBuffer,&eventBufferSize,
privateData,&numEvents);
if(rCode==ACSPOSITIVE_ACK)
{
if(eventBuffer.eventHeader.eventType == ACS_OPEN_STREAM_CONF)
{
printf("ACS_OPEN_STREAM_CONF message has been received\n");
}
else
{
printf("event type is incorrect...");
return;
}
}
else
{
printf("acsGetEventBlock failure...");
return;
}

/*-------------------------------------------------------------------
** Now that the stream has been successfully opened, go ahead and
** issue the make call function.
*/
rCode=cstaMakeCall(acsHandle,invokeID,caller,callee,
privateData);

/*-------------------------------------------------------------------
** Now we need to poll for events until a confirmation has been
** received that the PBX has been able to make the call. Note that
** if something is invalid, the program will enter an infinite loop,
** as this simple program just sits and waits until a confirmation
** returns that is successful.
*/
while (!done)
{
rCode=acsGetEventPoll(acsHandle,&eventBuffer,&eventBufferSize,
privateData,&numEvents);

if(rCode==ACSPOSITIVE_ACK)
{
if(eventBuffer.eventHeader.eventType == CSTA_MAKE_CALL_CONF)
{
printf("CSTA_MAKE_CALL_CONF message has been received\n");
done=1;
}
else
{
printf("event type is incorrect...");
}
}
}


/*-------------------------------------------------------------------
** All done! Time to close the stream now.
*/
rCode=acsCloseStream(acsHandle,invokeID,privateData);

if (rCode < 0)
{
printf("acsCloseStream failure...");
return;
}
else
{
printf("acsCloseStream success\n");
}

return;
}
```

Getting Help

Two magazines provide programming information for members of the Novell Professional Developers Program--*Bullets* and *Developer Notes*. Both of these will contain sample code, as well as information and TeleTips, a new program that will cover telephony tips.

In addition, a faxback service and a telephony forum on CompuServe will both be online soon (and both will include Teletips). And, of course, there is telephone support, where you can work directly over the phone with members of the Novell Developer Support group.

Future of the SDK

The future holds nothing but exciting advancements for NetWare Telephony Services. As the Telephony product is enhanced, the SDKs will be enhanced as well, so that everything that Telephony Services provides will be available for you to use in your applications. There will be support for other client platforms as well as new features for existing platforms. Also, as PBX drivers are introduced and enhanced, you will be able to take advantage of a larger set of the TServer API calls.

Keep your eye on *Bullets*, *Developer Notes*, FaxBack, and CompuServe for announcements on new releases. All in all, NetWare Telephony Services is an exciting developmental area involving integration of two very valable tools--the computer and the telephone.