SANGOMA

Application Note

Dialogic Diva Media Boards by Sangoma

Using the Dialogic Diva V-xPRI Boards in Modem Solutions

Executive Summary

The Dialogic Diva V-xPRI-Series Media Boards by Sangoma provide key functionality that can be used to build high-density modem banks with up to 240 ports in a single server. With a wide range of supported modem protocols, these boards are well suited for a variety of applications such as point-of-sale (POS) termination, monitoring, and termination of utility metering devices.

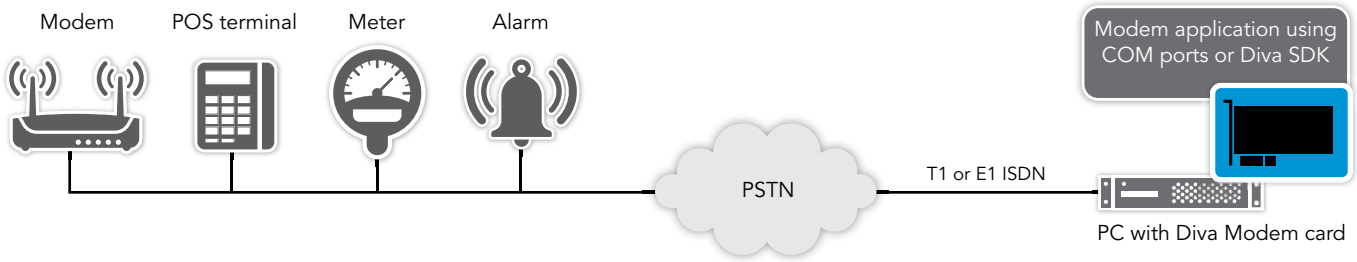


Figure A - High Level Solution Overview

This application note discusses two approaches to use the Diva V-xPRI Media Boards to deliver generic modem functionality in a solution. The two approaches are (1) to use the Modem AT Command Set or (2) to use the Dialogic Diva SDK. Commonly used license configurations and special deployment considerations are also described.

Table of Contents

Introduction

Most modem packages allow the modem to be controlled via either a proprietary API or the Modem AT Command Set. Similarly, the Diva V-xPRI Media Boards' architecture provides two major interfaces for controlling the modem:

- Modem AT Command Set
- The Dialogic Diva Software Development Kit (SDK)

Modem AT Command Set

This first interface option, which employs the time-tested, generic Modem AT Command Set, can be especially beneficial to users that already have a working application that is written to the Command Set. In such configurations, the application runs on the Diva V-xPRI Media Boards with no code changes, as long as the AT Command Set is from one of a number of broadly used modem chipsets. In Figure 1, the last column in green would be the path for a user choosing this option.

NOTE: Communication applications that use Microsoft's Telephony Application Programming Interface (TAPI) can be used to control the Diva V-xPRI Media Board's modem capabilities. However, while this functionality is available to configure in the Diva drivers, it is no longer supported by Dialogic.

Diva SDK

The Diva SDK offers three Diva APIs that allow the use of different programming paradigms (languages and environments), such as C, C#, .NET, or VB.Script. Solution vendors and application developers that use the Diva SDK find a wide range of tools, sample applications, developer technical support, as well as on-going fixes and enhancements to facilitate solution development, deployment, and maintenance.

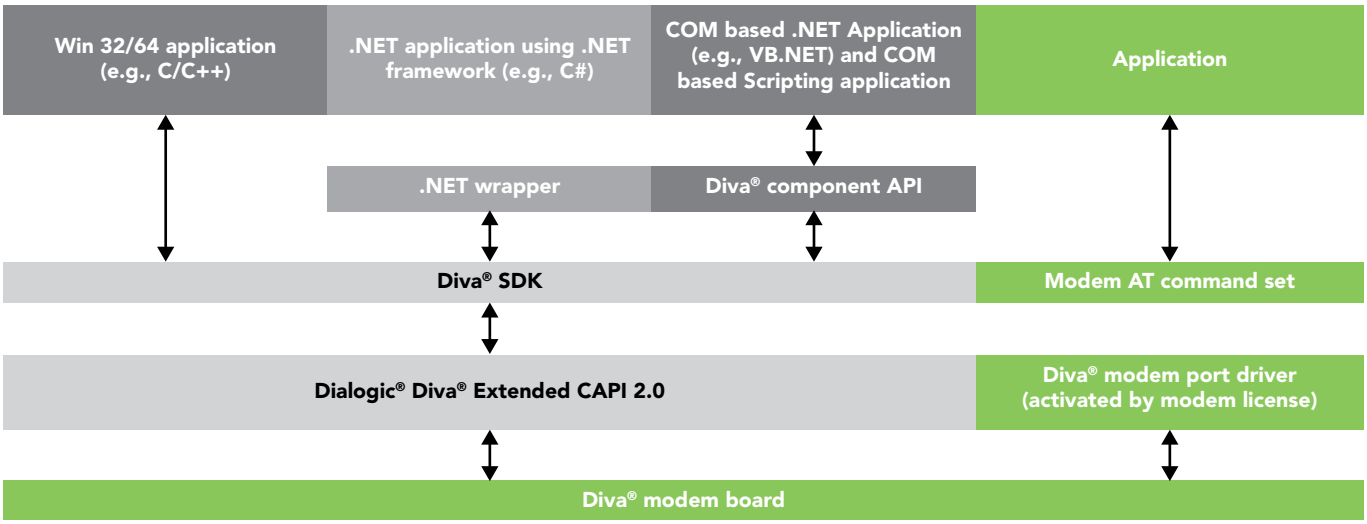


Figure 1 – Architecture Diagram

Diva Modem Boards Overview

Board Features

The Diva V-xPRI are high-density (multi-channel) boards with digital (T1 and E1) interfaces. The Diva V-xPRI Media Boards are available in 1, 2, 4, and 8 E1/T1 variants.

- The Diva V-1PRI and Diva V-2PRI are half size PCIe boards.
- The Diva V-4PRI board comes in two versions: a half-length (12 DSPs) PCIe board and a full-size (24 DSPs) PCIe board.
- The Diva V-8PRI is a full size PCIe board.

Supported by both Windows and Linux, the Diva V-xPRI Media Boards enable COM port or TTY port emulation in the operating system, which allows transparent support of existing modem and COM port applications.

Support for V.22FastConnect for POS applications, V.22bisFastConnect, and V.29FastConnect minimizes training and connect time to beneficially increase overall data acquisition and transfer rates per channel.

Regardless of the programming API that is being used, the Diva V-xPRI Media Boards require that modem licenses be purchased from Dialogic and applied separately to enable modem functionality. Licenses can be purchased on a one-license-per-channel basis for as many channels as are required. The licenses are associated with a specific Diva V-xPRI Media Board. The Diva V-xPRI Media Board and associated license can be moved to another server, should there ever be an issue with the original server.

Option 1 – Using the Modem AT Command SET

Modem AT Command Sets can be used as a command language to handle modem calls, and the Diva "modem port driver" is used to enable this functionality. In this case, the Diva V-xPRI Media Board is made to behave like a "generic" modem as long as a modem license has been applied to the board.

Configuring the Diva Board for Modem Port Driver

After the modem license has been applied, the first configuration step is to add the modem service in the Dialogic Diva Configuration Manager:

1) Right-click on the Services line, and click **Insert Modem Pool**. "Pool" implies that more than one modem channel needs to be enabled. An icon labeled "Analog" is placed on the Services line.

2) Select the **Analog** icon, and on the right pane, set the desired number of modems. This can never exceed the number of applied modem licenses.

3) Right-click the **Analog** icon, and then click **Select for Binding**.

4) Right-click the greenish icon (labeled **V-1PRI**, **V-2PRI**, **V-4PRI**, or **V-8PRI**) on the Boards line, and then click **Complete Binding**. If the binding is successful, a line will now connect the **Analog** icon and the board icon. Repeat this step for each TDM port on the board.

5) From the **File** menu, click **Activate** for the changes to take effect. This could take a few seconds depending on the number of modems configured.

6) When prompted, restart the computer. If the addition of the modems is successful and the computer restarts, the modems will be listed in the Windows Device Manager > Modem directory.

These steps provide the entry point for the modem application into the Diva modem port driver. The final configuration will look similar to Figure 2.

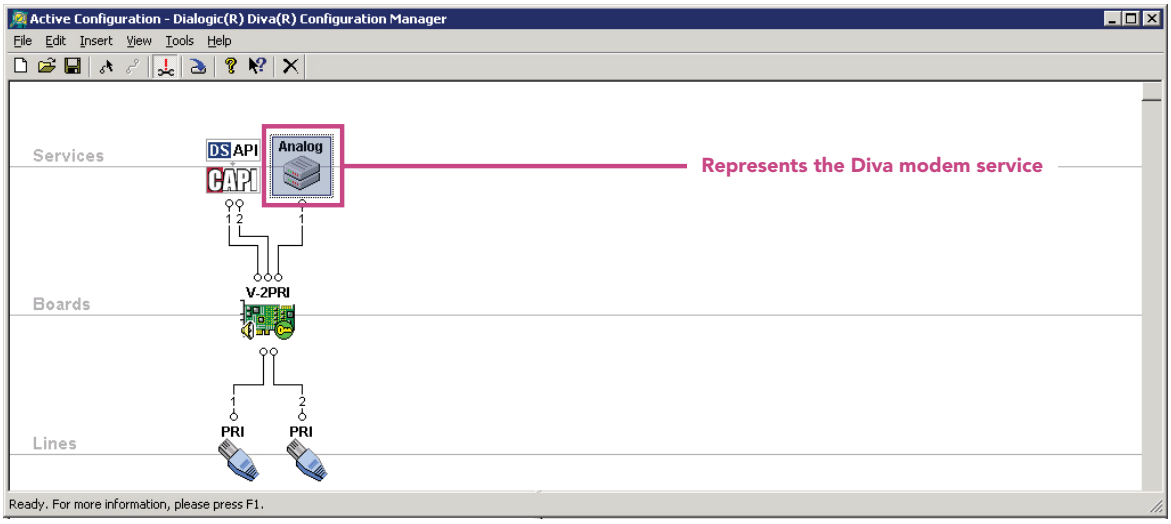


Figure 2 – Diva Modem Service

Configuring an Extra Modem Initialization String

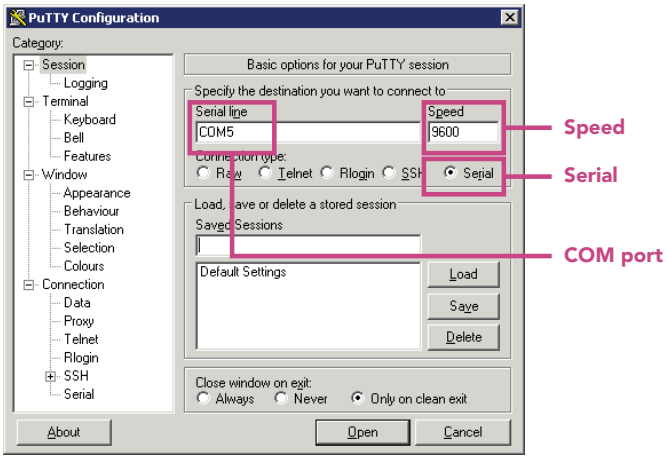
Default modem initialization strings can be easily set on a modem-by-modem basis under the advanced modem properties in the Windows Device Manager > Modem directory.

1) Expand the Modem directory to view all individual modems. The total number of modems should be the same as enabled in the Diva Configuration Manager.

2) Double-click each modem, click the **Advanced** tab, and enter the desired "Extra initialization string." Note that this is called "Extra" because the port driver will send the default initialization string according to the default modem profile. The most common extra initialization string is to set the modulation. Refer to the following example.

```
AT+MS=V34
```

Programming the Diva Board for Modem Port Driver



The "Supported TTY Profiles" section of this same Reference Guide provides information on the TTY profile and determines the first burst of AT commands that get issued. Users may issue the commands from their applications, but for quick test purposes, an Open Source SSH client such as PuTTY or Tera Term can be used.

If using PuTTY, the initial configuration upon application launch only requires three settings: changing the connection type to "Serial", changing the COM port number to the same COM port number indicated in the Diva Configuration Manager, and entering the desired or expected modem speed. Refer to Figure 3.

Figure 3 – PuTTy Usage Example

After entering the three settings, click **Open** to get the interface ready and to open a window to accept AT commands.

First AT Commands Sent

The default profile is #5 (Asynchronous modem up to V.90 with full negotiation and V.42/V.42bis/MNP), which sends the following initialization string to the port driver.

```
at&F5E0V1Q0&D2&C1S0=0
```

The initialization string comprises these seven AT commands.

&F5 ⇢ Set profile #5

E0 ⇢ Echo mode OFF

V1 ⇢ Plain text result codes

Q0 ⇢ Return results codes

&D2 ⇢ Data terminal ready options

&C1 ⇢ Of no effect, but accepted for compatibility reasons

S0=0 ⇢ Disable auto-answer

The AT commands are sent unless the application overwrites them. Note that the commands are not case sensitive. For an application or user to overwrite the default commands, send another string that begins with "AT&F". Refer to the following example.

```
at&f9
```

This command tells the port driver to overwrite the default profile and use the HDLC Transparent protocol stack for PPP instead. If the command does not begin with "AT&F", the port driver will assume the command is additional to the default string. In this case, the value of "9" for PPP is obtained from the Supported TTY Profile list as follows.

Profile	Description
14	Autodetection of B-channel protocol.
1	X.75/Transparent/Transparent protocol stack. Data compression in accordance with V.42bis is detected automatically for incoming calls.
2	V.110 synchronous mode.
3	V.110 asynchronous mode.
4	Synchronous modem with V.42/V.42bis.
5	Asynchronous modem (up to V.90) with full negotiation and V.42/V.42bis/MNP.
6	V.120, 64 kbps, V.42bis compression auto-detection for incoming connections.
7	V.120, 56 kbps, V.42bis compression auto-detection for incoming connections.
8	Bit-transparent access to B-channel data [a].
9	HDLC/Transparent/Transparent protocol stack that is widely used for PPP connections [b].
10	Same as profile 9, but with 56000 bps.
11	BTX.
12	BTX.
15	X.75 with data compression in accordance with V.42bis. This profile should be used for outgoing calls if you wish to use data compression in accordance with V.42bis. If the opposite side does not support data compression, the connection will be established without data compression.
16	PIAFS 1.0 32 kbps [c].
17	PIAFS 2.0 64 kbps [c].
18	PIAFS 2.1 32/64 kbps [c].

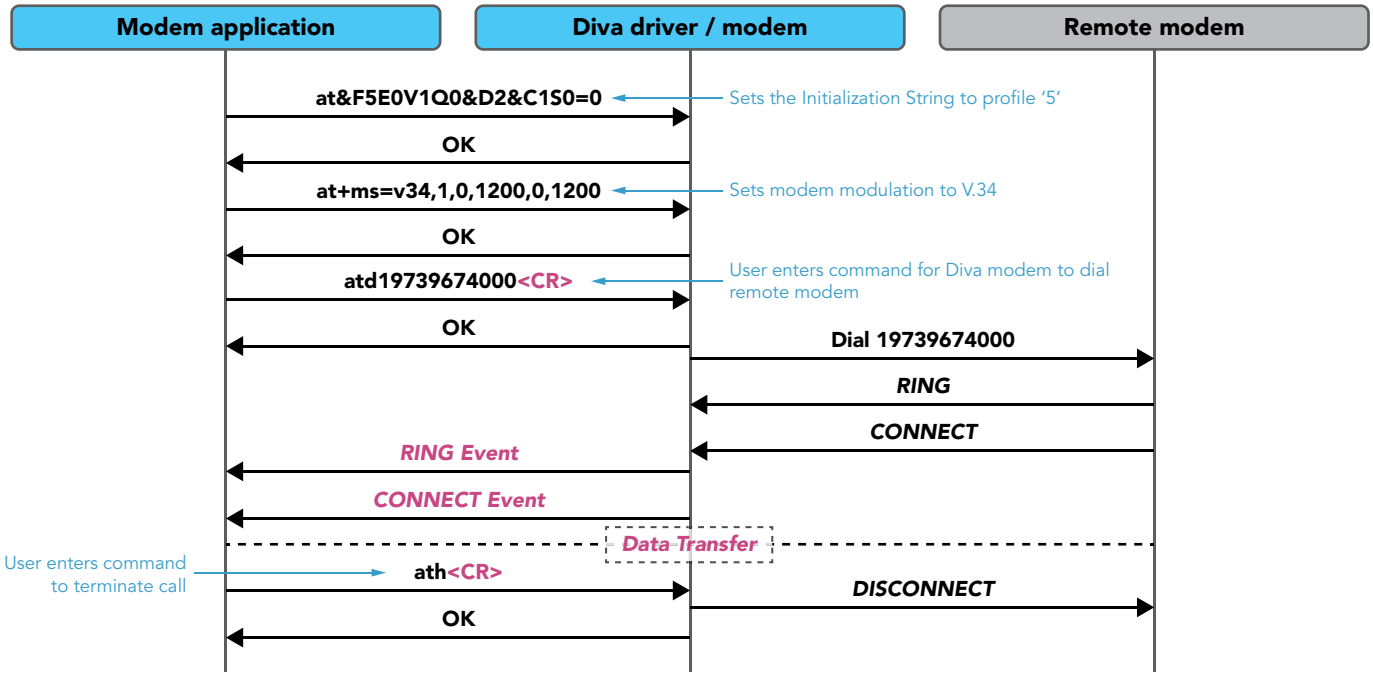
Second AT Commands Sent

The second AT commands are found under the **Windows Device Manager > Modems > Properties > Advanced** tab. Typically, this sets the modem modulation. For example, the following command sets the modulation and includes the lower and upper speed limits for upload and download data.

```
at+ms=v34,1,0,1200,0,1200
```

Call Connection

Basic call connection is dialed using the ATD command, where "D" is the dial command. Command switches such as 'T' for tone dialing or 'P' for pulse can be appended to the ATD. If the dialing progresses as expected, the local modem reports a RING, while the remote modem issues an ATA command, which is the AT command and "A" for Answer. If the V1 command had been included, as in the previous sample initialization (**F5E0V1Q0&D2&C1S0=0**), the connection result that includes the modulation is returned in plain text and appended to the CONNECT output. Once in the connected state, both modems are ONLINE or in DATA TRANSFER MODE. In DATA TRANSFER MODE, any character entered at one terminal is automatically transferred to and appears at the other terminal. A typical call flow can be represented as follows.



Eventually, one side issues an ATH command (where the "H" stands for hang up) to terminate the call.

Option 2 - Using the Diva SDK

Configuring the Diva Board

The Diva APIs are used for programming the Diva modem. The Diva API provides control and flexibility to programmers because they can adjust the business logic as required to fulfill a particular solution set. The Dialogic Diva Hardware Installation Guide provides an overview of how to install and connect the Diva V-xPRI Media Boards. The Dialogic Diva Media Board drivers also need to be installed on the server. These Diva drivers can be downloaded from the Dialogic website (free registration required). If the modem application will be using only the Diva SDK or CAPI, then only the Diva Server API and CAPI services (DSAPI/CAPI) icon need to be present on the Services line of the Diva Configuration Manager. An example of the configuration screen is shown in Figure 4.

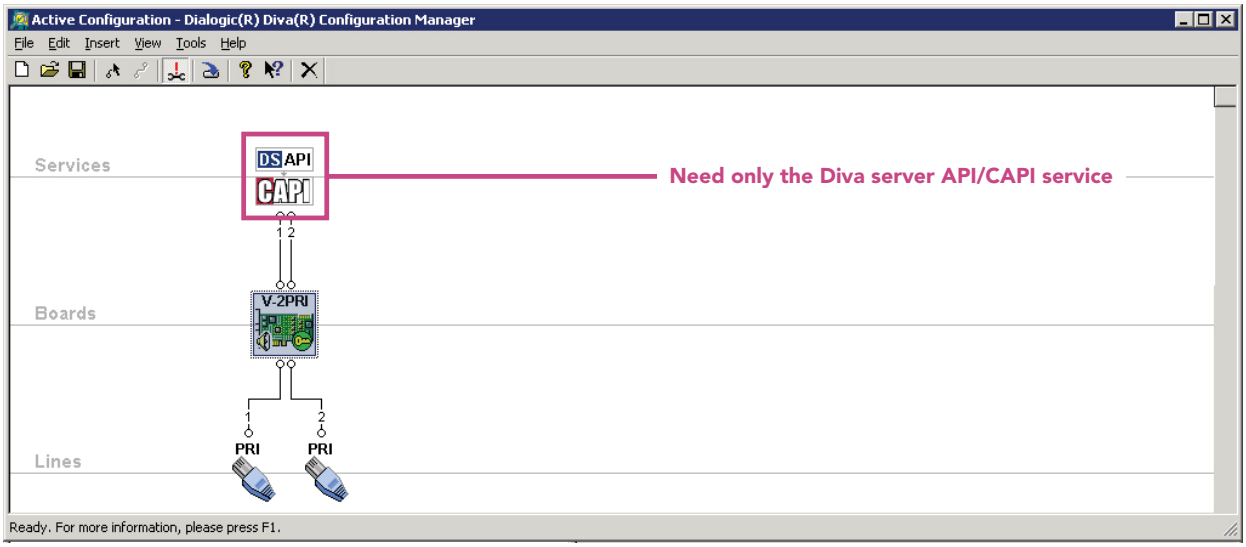


Figure 4 –DSAPI/CAPI Service Configuration

Note:

- All modem initialization will be done using the Diva API's "Diva call properties."
- Modem initialization strings via the Windows Device Manager are not necessary when using the Diva API and will be ignored if configured.

Programming the Application

A general overview of programming with the Diva API is described in the *Dialogic Diva API Developer's Reference Guide*. However, when programming the Diva modem, there are some call property settings that need to be specifically programmed for the desired modem protocol. The settings are the equivalent of the modem port driver initialization strings. A summary of the program tasks are as follows:

1) Initialize the Diva SDK.
2) Register the application to the Diva SDK.
3) Wait for the call on all channels.
4) Call the event handler.
5) Define the call modulations.
6) Answer the call.
7) Retrieve the modem connection properties.
8) Receive the data.

The following code snippets demonstrate these steps. The code snippets are from a sample application called "POSModemScenarios" that comes with the Diva SDK package and demonstrates a typical POS modem scenario. The full source code can be found in the SDK package's *~\Basic\samples\C\POSModemScenarios* directory.

Initialize the Diva SDK

```
if (DivaInitialize() != DivaSuccess){
            return -1;
}
```

Register the Application to the Diva SDK

```
if ( DivaRegister ( &hApp, DivaEventModeCallback, (void *) CallbackHandler, 0, 1, 7, MAX_FRAME_SIZE )
!= DivaSuccess ) {
            return -1;
}
```

Wait for the Call on All Channels

```
if ( DivaListen ( hApp, DivaListenAll, Device, NULL ) != DivaSuccess ){
 return -1;
}
```

Call the Event Handler

In this example, the event handler is called because an incoming call has already been detected.

```
void CallbackHandler (DivaAppHandle hApp, DivaEvent Event, PVOID Param1, PVOID Param2) {
 const char* pModemType;
 BOOL bSet = TRUE;
 DWORD dwSet = 0;
 DWORD ConnectedNorm;
 DivaCallInfo Info;
 DWORD BytesRead;
 DWORD Result;
 switch (Event){
 case DivaEventIncomingCall:
 // Code to handle the call
```

Define the Call Modulations

This is a continuation from the callback handler snippet in the previous example. This step does not have to be performed until an incoming call has been detected. Once the call type is defined as "modem," most modulations are enabled by default; therefore, it is the application developer's responsibility to comprehensively disable all modulations that should not be used for the particular call.

In the following example, Bell 103, Bell 212A, and V.22bis FS are the desired modem types. Bell 103 and Bell 212A were already enabled by default; therefore, they do not need to be enabled. The following modulations were also enabled by default, and should be explicitly disabled for this example: V.21, V.23, V.22, V.22bis, V.32, V.32bis, V.34, V.90PCM, ALLFS, K56Flex, and X2. The last API function call highlighted in the example below explicitly enables V.22bis FS, while all prior statements disable default features that are not properties of the three desired modem types.

```
case DivaEventIncomingCall:
if ( hSdkCall == NULL ){
hSdkCall = Param1;
pModemType = GetModemScenarioString ( ModemScenario );
switch ( ModemScenario ) {
 case 0: // Bell 103, Bell 212A (async or sync), V22bis (async or sync), V.29
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV21,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV23,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV22,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV22bis,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV32,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV32bis,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV34,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationV90DPCM,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationAllFS,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationK56Flex,DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableModulationX2,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
DivaSetCallProperties(hSdkCall,DivaCPT _ EnableModulationV22bisFS,DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
break;
```

Other call properties, such as Answer Tone Duration, that do not have to do with modulation are then set.

```
if ( AnswerToneDuration == 0) {
DivaSetCallProperties(hSdkCall,DivaCPT _ DisableAnswerTone,(DivaCallPropertyValue*) &bSet, sizeof ( BOOL ) );
}
else if ( AnswerToneDuration != -1 ){
DivaSetCallProperties(hSdkCall,DivaCPT _ AnswerToneDuration,DivaCallPropertyValue*) &AnswerToneDuration, sizeof (
DWORD ) );
}
if ( ConnectTimeout ){
DivaSetCallProperties ( hSdkCall, ivaCPT _ ConnectTimeout,DivaCallPropertyValue*) &ConnectTimeout, sizeof ( DWORD ) );
}
```

Answer the Call

The call can be answered with either DivaAnswerModem() or DivaAnswer() but with call type set to "modem," as illustrated below. In the next three code snippets, the key API events and functions involved in getting the call to a connected state and retrieving incoming data are highlighted.

```
DivaAnswer ( hSdkCall, hMyCall, DivaCallTypeModem );
printf ( "Answer incoming call for '%s'\n", pModemType );
```

Retrieve the Modem Connection Properties

```
case DivaEventCallConnected:
memset ( &Info, 0x00, sizeof ( Info ) );
Info.Size = sizeof ( Info );
DivaGetCallInfo ( hSdkCall, &Info );
DivaGetCallProperties ( hSdkCall, DivaCPT _ ConnectedNorm, (DivaCallPropertyValue*) &ConnectedNorm, sizeof
( DWORD ) );
printf ( "Connected, Norm '%s', Rate %d/%d \n", GetConnectedNormString ( ConnectedNorm ), Info.TxSpeed,
Info.RxSpeed );
PrintConnectedModeString( hSdkCall );
break;
```

Receive the Data

```
case DivaEventDataAvailable:
printf("Data Available %d bytes\n", (DWORD)(unsigned long) Param2 );
/*
* Read all data from the Diva API, otherwise this event will not
* be indicated again.
 */
do{
Result = DivaReceiveData ( hSdkCall, DataBuffer, sizeof ( DataBuffer ), &BytesRead );
if ( ( Result == DivaSuccess ) && BytesRead ){
printf ( "Read %d Bytes received data\n", BytesRead );
 }
} while ( ( Result == DivaSuccess ) && BytesRead );
break;
```

Deployment Considerations

Adapting to Different Modem Modulations

The most frequent challenge that arises when implementing a modem solution is being able to adapt to different modem modulations when the remote modems are different makes and models. Whether the AT Command Set or Diva API is being used, the Diva V-xPRI Media Boards support "automoding," which enables the remote modem's modulation to be adjusted accordingly.

Some older legacy modems might not be designed to easily adjust modem attributes. In order to determine the correct modem modulation for such a legacy modem, the user may have to test various modulations or, when possible, obtain information on the original initialization strings for the modems. Another common situation encountered is the need to set up a new telecommunication network path when users build a new server using the Diva V-xPRI Media Boards. In this case, it is recommended to perform a basic connectivity test to verify the modems are indeed reachable by the new network path.

Using Third-Party Virtual Modem Software

Third-party solutions that are able to assist in virtualizing modem servers can be useful when deploying solutions in the field. An example of one such application uses Dialogic Diva V-xPRI Media Boards and a virtual modem solution from Tactical Software, which provides ease of use and flexibility in dealing with a large number of different types of modems. As of April 2016, a description of the "TACServe" application can be found on the Tactical Software website.

The TACServe application has two components: the server that runs on the Diva modem server and a client that runs on the agent computers. The TACServe application can be useful when there is a requirement for remote modems of different types and modulations. It gives users the ability to set up different initialization strings, which may be required by the different modem types.

Figure 5 illustrates an example topology where the TACServe software was used to create virtual modems on an agent's computer.

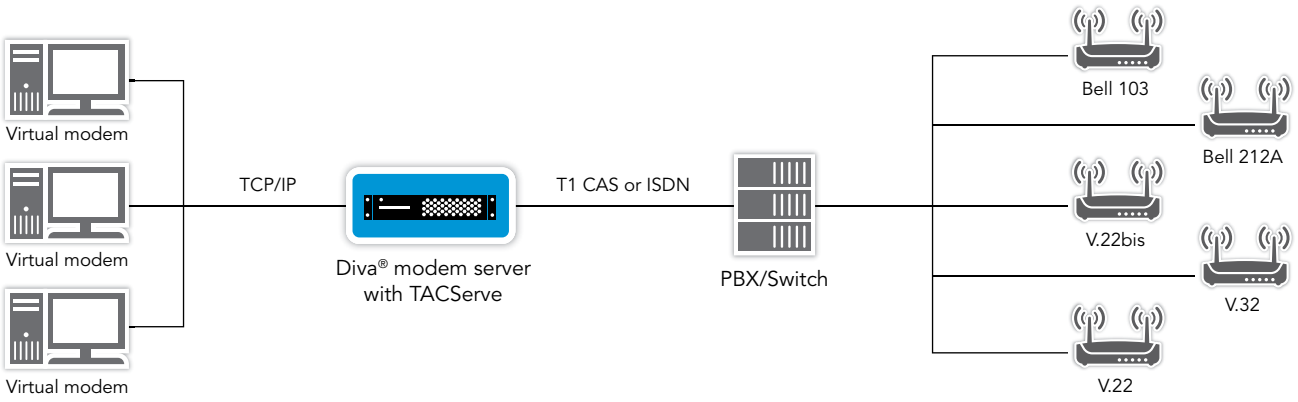


Figure 5 – TACServer Network Configuration Sample Overview

In this solution, the Diva modem inf file was used to create the virtual modems on the agent's computers. Connection items with customized initialization and dial strings can be created on the agent computers for each remote modem to which an agent will be connecting.

It is recommended to test the connections. If testing reveals issues with the connections, adjustments to the modem initialization strings may be needed to establish a suitable connection. The client application in Figure 5 above uses the virtual modem to connect to the TACServe server on the Diva modem computer using TCP/IP and sends AT commands with the customized initialization and dial strings. Then, the TACServe server can send those commands to the Diva modem for dialing out to the telephone network and the remote modems.

WHY SANGOMA?

Sangoma's customer-centric approach, product innovations, and worldwide network of distribution partners give you the industry's best-engineered, highest quality, IP and UC solutions, supporting "any app, anywhere" for businesses and service providers of all sizes. All Sangoma products are backed by more than 30 years of IP communications experience, expert engineering and technical resources, as well as a comprehensive warranty

Training and certification programs to set yourself apart from your competition.

Sangoma is a single source solutions provider. We manufacture our solutions, so they work seamlessly.

We offer several opportunities for reoccurring revenue.

Best in class discounts for partners

Sangoma Technologies
100 Renfrew Drive, Suite 100 Markham ON L3R 9R6 CANADA
1 800 388 2475 toll free in N. America
+1 905 474 1990 international direct
www.sangoma.com
sales@sangoma.com

SANGOMA TECHNOLOGIES