CT Application Development Environment

User Guide

January 2002

Order Number: 05-1309-001

Contents

Contents

Preface

This chapter describes how this book is organized and contains the following sections:

- How to Use this Guide
- Documentation Suite
- Hardware and Software Requirements
- Contacting Us

How to Use This Guide

This section explains what you need to know before using CT ADE and describes the content and layout of this guide.

Required Knowledge

To use CT ADE, you need to be familiar with:

- Windows NT or Windows 2000 operating system
- Computer telephony terminology and concepts
- Computer telephony hardware
- If you plan to use CallSuite, you need to know one of these programming languages: Visual Basic, Visual C++, or Delphi.

Format Conventions

The following conventions should help you navigate more easily through this guide.

Displaying Code Examples

Text in Courier font indicates computer code, for example:

```
program

   vid_write("I am a VOS program!");

   vid_write("Type any key to exit...");

   kb_get();
```

```
    vid_write("Exiting now.");

    exit(0);

endprogram
```

Navigating Application Menus

When describing which submenu item to choose from a higher-level application menu, the pipe symbol (|) is used. For example: From the Start menu, choose Programs | Parity Software | CallSuite Help.

Introducing New Terms and Showing Audible Messages

Text displayed in italicized style is used to introduce a new word or concept. Subsequent use of the term appears in normal text.

Italicized text is also used to demonstrate that a Computer Telephony application is playing an audible message. For example:

"Press 1 to hear new messages, press 2 to hear saved messages, press 3 to..."

Identifying Text That You Need to Type

Text that you need to type usually shows up in a sequence of steps and specifies exactly what you need to type in order to complete the step. It displays in either bold style or Courier font.

For example, in the following sentence you type the word "inbound" in the field specified or on the command-line described:

1. "Name the file **inbound**, and make sure the Add to Project box is checked."

Code that you need to type can also display in Courier font. The following step shows an example of this:

2. Click in the Editor window on the right and type the following code into it:

```
program
    TrunkWaitCall();
    TrunkAnswerCall();
    if (TrunkGetState() strneq "Connected")
        voslog("Unexpected trunk state ", TrunkGetState());
        stop;
endif
InboundCall();
TrunkDisconnect();
restart;

endprogram
```

Providing Tips, Cautions, and Warnings

Text beneath these icons provides information to help you improve your methods or avoid mishaps.

 Tip:

A lifesaver Tip offers helpful hints or alternative ways of doing tasks.

 *Caution:*

Text beneath a Caution sign describes how to avoid an annoying situation, such as a common programming mistake or receiving an error dialog.

 **Warning:**

Text beneath a Warning sign explains how to avoid a catastrophic situation, such as losing data or crashing your hard drive.

Documentation Suite

CT ADE comes with a variety of documentation, both online and printed, to help you use the product.

Table 1.P CT ADE Documentation Suite

Online Help Files	Printed Documents
Graphical VOS User Guide	Installation Instructions
RLL help files	CT ADE User Guide (this guide)
CallSuite User Guide	
SimPhone Help	
NetHub Plus User Guide	
Sample Application ReadMes	
WaveTest	
Audiotst.exe ReadMe	
Media Toolbox (Utilities help files)	
License Upgrade Help	

Online Help Files

When you install CT ADE, menu items are created on the Start menu that take you to the latest online help files. We strongly recommend that you spend some time exploring the online help to see what information is available. Investing a few hours to do this will repay itself many times over when you begin developing CT systems.

Graphical VOS User Guide

The Graphical VOS online help file describes what the Graphical VOS development environment is and how to set it up for use. It also describes what Topaz is and how it works with Graphical VOS. You can find extensive information about using VOS script and the FlowCharter including tutorials to help you begin developing your applications. There is also a tutorial on using the Debugger tool.

The online help file also includes tables that display Topaz RegIDs for Get/Set functions (RegIDs values are used internally by Topaz) so you can retrieve information about the system or issue a Technology-specific command at runtime.

The help file contains information about the Runtime Link Libraries (RLLs) available with Graphical VOS that you can use to add additional functionality to your applications.

Finally, you can find information on logging and tracing, troubleshooting problems, and contacting Technical Support.

RLL Help Files

The Graphical VOS development environment includes online help for each of the following RLLs:

- ADO RLL—Provides VOS applications with direct access to Microsoft ActiveX Data Objects (ADO). This RLL is documented in the Graphical VOS Online User's Guide.
- ATM RLL—Lets you share resources on the SC bus between PCs on an ATM (Asynchronous Transfer Mode) network.
- Dialog RLL —Allows you to include Windows dialogs in your VOS applications to provide GUI interaction with the screen/keyboard user of the VOS system.
- Fixed Point Math RLL ReadMe—Provides functions to add, subtract, multiply, and divide decimal numbers from within VOS programs. This RLL is documented in the Graphical VOS Online User's Guide.
- Sockets RLL—Lets you send and receive messages to VOS tasks running on remote PCs connected via an IP network.

- Web RLL—Has functions that enable your VOS program to: get documents from a Web server, access data from active Web pages (for example, from an e-commerce server), and send and receive files via FTP.

CallSuite User Guide

The CallSuite online help file tells you how to set up the CallSuite development environment and describes the functions of the various components contained in CallSuite. Also described are the methods, properties, and events belonging to each component and details about how to use them. Additionally, you will find information about Topaz and how it works with CallSuite.

To help you get started creating an application using the CallSuite Wizard, we've included a tutorial as well as information about logging and troubleshooting problems, and contacting Technical Support.

SimPhone Help

The SimPhone online help file describes how to use SimPhone and includes information about compatibility with CallSuite and Graphical VOS.

NetHub Plus User Guide

The NetHub Plus online help file contains information about installing, configuring, and using the NetHub Plus ActiveX control. It also includes a tutorial, RLL reference, and information about NetHub Plus data types, methods, properties, and events.

Sample Application ReadMes

The sample applications are provided to give you a few examples of the kinds of applications you can write and you are encouraged to copy and modify these samples for your own use. Each sample has a ReadMe that describes what the application does and how it. If you are using Graphical VOS, you can view both FlowCharter and Script samples (samples written in the VOS programming language). If you are using CallSuite, you can view sample applications in Delphi, Visual C++, and Visual Basic. Sample applications are automatically included in your installation and you can access them from the Start menu.

WAVETEST Help

The WAVETEST online help file accompanies WAVETEST, which a command-line utility that lets you manage Wave devices and files. The help file describes how to display installed Wave devices, test your system for supported Wave formats, and display the a Wave file's format. This file also includes information to help you troubleshoot problems.

Audiotst.exe ReadMe

The Audiotst.exe ReadMe describes how to use this application to find out the number of voice boards present on your system.

Media Toolbox

Media Toolbox is a collection of ActiveX controls you can use to further manipulate telephony sound files.

License Upgrade Help

The License Upgrade online help explains how to use the License Upgrade utility to upgrade your existing license.

Printed Documents

Installation Instructions

Included in your CT ADE package, this document tells you how to install CT ADE.

CT ADE User Guide

The printed User Guide (this guide), describes the two development environment and various components that make up CT ADE. See the sections titled “Chapter Contents” for a detailed list of contents.

Hardware and Software Requirements

This section gives you a general overview of hardware and software requirements you need to run CT ADE. However, depending on whether you are running CT ADE in evaluation mode or development mode, and what hardware, drivers, and speech engines you have installed, etc., your requirements will vary.

Meeting Hardware Requirements

To run CT ADE, you need the following hardware:

- Telephony Hardware and Software

  Refer to the documentation provided with your board(s) for information on hardware installation. If you are running in evaluation mode, you may not need telephony boards or drivers; SimPhone can be used for development and testing of many runtime applications.

- Hardware Key

  If you plan to run CT ADE in development mode, a hardware key needs to your PC. This key, which may be included with your installation materials, is programmed with information about your license. It has a standard

parallel port connector and USB that allows normal operation of printers and other devices that can be attached to the port.

You do not need the hardware key if you are running CT ADE in evaluation mode.

- How many lines can my PC Run?

  A commonly asked question is how much hardware (processor speed, RAM, etc.) is required to run a given number of telephone lines. The answer depends on many factors relating to your particular application. Your distributor can advise you about specific information for your particular configuration.

  You do not need any phone lines if you are running in evaluation mode—you can use SimPhone instead (unless you are evaluating Text-To-Speech, Voice Recognition, or Faxing capabilities; in this case you'll need the appropriate board/speech engine). For more information about SimPhone, see the chapter titled "Using SimPhone."

Meeting Software Requirements

To run CT ADE, you need the following software:

- PC running Windows NT or Windows 2000

  For more information on Dialogic cards, see your distributor. For information about CT Media, see the online help.

- Topaz, which is automatically included in the CT ADE installation

  For information about configuring Topaz, see the online help.

- SimPhone, which is automatically included in your CT ADE installation

  You only need SimPhone if you do not plan to use any telephony hardware.

- To use CallSuite, you need access to a programming language you are familiar with, such as Visual Basic, Visual C++, or Delphi.

- Third-party applications

  To develop certain applications that use third-party products and to run certain sample applications, you need additional third-party software such as Java™ Development Kit (JDK) or SpeechWorks. The sample application ReadMes, Graphical VOS, or CallSuite online help list the requirements in these cases.

Contacting Us

We would like to hear from you! Please e-mail us with any comments and suggestions you have on the CT ADE product, including the documentation. You can send us an e-mail one of two ways:

- E-mail us at ade_documentation@intel.com.
- E-mail us using the auto-feedback feature in the online help.

CT ADE Overview 1

This chapter describes the CT ADE suite of products:

- CT ADE Overview
- Topaz
- Application Development Environments
- Additional Tools and Utilities

CT ADE Overview

Computer Telephony Integration (CTI) is an implementation of telephone and computer technology that enables voice and data processing equipment to work together, letting you efficiently and easily exchange information via tools such as telephones, computers, faxes, and voice recorders.

We've utilized this technology to develop CT ADE, which stands for computer telephony (CT) application development environment (ADE). CT ADE is made up of the following components: the Topaz engine, the Graphical VOS programming environment, and the CallSuite programming environment. Together these components provide an environment in which you can develop, test, debug, and run your own CT applications. CT ADE also communicates with and utilizes the API drivers, telephony boards, and any additional speech engines already installed on your system to perform CT functions.

CT ADE offers development tools and a choice of two programming interfaces—CallSuite or Graphical VOS, that eliminate the need to write directly to the API in C or C++. Additionally, the underlying Topaz architecture supplies an abstraction layer that sits on top of an API and executes low-level CT functions so you don't have to worry about differing hardware and API protocols. The Topaz commands you use to initiate functions are simple to use, easy to learn, and consistent.

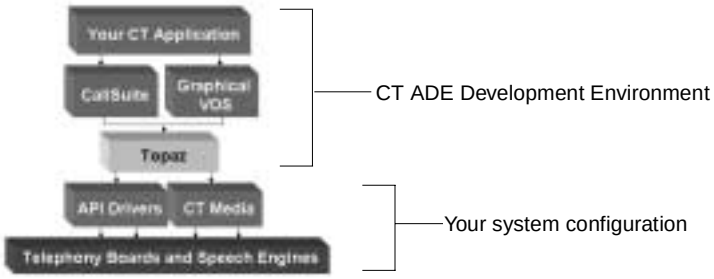


Figure 1-1 CT ADE development environment and system configuration correlation

Before installing CT ADE, you must first add any phone lines, fax machines, etc., that you plan to use in your telephony application. Next, you install the APIs for the drivers, and finally you install CT ADE.

Figure 1-2 Direction flow of system installation

Following installation, you run a scanner utility that creates a database and populates it with configuration information about any telephony boards and speech engines installed on your system. At this point you can begin developing your CT application, which references the database via Topaz to carry out the CT functions you request using commands called functions in Graphical VOS and methods in CallSuite.

Figure 1-3 Direction of application flow

The commands are easy to use regardless of trunk type or API. For example, if you invoke the VOS function *TrunkAnswerCall*, the Topaz layer determines which trunk interface is being used and which API function to use: *dx_sethook* (R4 analog), *dt_setsig* (R4 T-1), *cc_Answer* (R4 PRI), or *CTscr_AnswerCall* (S.100 / CT Media). We are committed to keeping Topaz applications portable to different APIs and trunk types as the industry evolves and customer needs change.

Topaz

The design of Topaz builds on more than a decade of experience in dealing with telephony APIs, cross-platform portability, and the wide variation in capabilities offered by telecommunications hardware and services. Topaz mediates between your programming commands and any telephony boards and speech engines installed on your system in order to carry out the commands you type.

The most important feature of Topaz, however, is API Transparency. Topaz relieves you from dealing with many chores that native APIs require, such as complex API function signatures including handles, bit-fields, pointers and structures. Topaz also determines the installed hardware and trunk configuration. Additionally, the same set of functions and methods work under all supported telephony APIs (Dialogic R4, S.100 / CT Media etc.) and all supported trunk types (analog loop-start, T-1 E&M, E-1/R2, PRI, etc.). Compared with C programming, this means that typical application functionality such as making or receiving calls, or playing menus and prompts, is substantially easier to create and maintain.

Finally, the API transparency Topaz provides lets you avoid locking yourself into committing to a particular API. Unlike a C/C++ program, your Topaz application supports portability to different APIs and trunk types as the industry evolves and your end-user needs change.

Evaluation and Development Modes

After installing CT ADE, you choose an ADE (CallSuite or Graphical VOS) that you run in either evaluation mode or development mode. With a valid hardware key attached to your PC, the software runs in full development or runtime mode. Without the hardware key, the evaluation mode runs.

Evaluation Mode

The following restrictions apply when you run in evaluation mode:

- Your applications cannot control telephony boards so you must use SimPhone to simulate making or receiving a phone call.
- Applications that were compiled in Evaluation mode must always use SimPhone, even if a runtime hardware key is later attached to the PC.

Development Mode

When you attach a valid hardware key to your PC, Graphical VOS and CallSuite can simultaneously run up to two instances of each of these Resource types: Media Resources, Fax Resources, Conference Resources, Text-To-Speech Resources, and Voice Recognition Resources.

Applications compiled in development mode run on as many API/driver combinations (called Resources) that the attached development or runtime key will allow. For example, if you wanted to run an application with 64 Media Resources, 30 Conference Resources, and 64 Fax Resources, you would need to attach a 64-port hardware key to the PC.

For more information about Resources or the hardware key and how to configure it, see either the Graphical VOS or CallSuite online help.

Application Development Environments

CT ADE offers you a choice of two environments in which to develop your applications: Graphical VOS or CallSuite. Each environment has been designed with unique advantages and one is sure to suit your needs.

Graphical VOS

You can create and deploy telephony systems with Graphical VOS, a complete development environment that includes Topaz, the VOS Language Compiler (VLC) and runtime engine, FlowCharter, Editor, Runtime Link Libraries, Debugger, and SimPhone.

For information about SimPhone, see the section titled, "Additional Tools and Utilities" in this chapter.

VOS Language Compiler and Runtime Engine

With the VOS programming language you can write and maintain standalone computer telephony (CT) applications. Similar looking to C, VOS was specifically designed to support CT technology. The coding process is simpler than coding in C or C++ because much of the application details are handled by the VOS engine, which also make maintenance and modifications easier to manage.

VLC translates VOS source code into a VOS executable at runtime, which can then be used by the VOS runtime engine.

Graphical VOS provides an environment in which you can write, test, debug, and run your CT applications in a Windows environment. The VOS engine is responsible for executing the applications built with the VOS Language Compiler.

FlowCharter

VOS FlowCharter is a graphical tool that lets you create and edit telephony applications. Instead of writing source code, you draw telephony applications by arranging cells (modules of code that perform specific CT functions) into a flowchart layout and linking them in the order you want the application to flow. VOS FlowCharter is also ergonomically designed so you can create applications quickly making relatively few mouse movements and clicks. To see the tutorial showing you how to use FlowCharter, see the chapter titled, "Using Graphical VOS" in this guide.

Editor

The Graphical VOS Editor is a syntax-highlighting editor designed specifically for VOS language source code. As you edit code, the Editor highlights and colorizes different VOS elements, making it easy to follow the flow of your code. You can use the Editor to create new VOS source and function files, or to edit existing ones.

Runtime Link Libraries

Runtime Link Library (RLL) add-ins are extensions that expand VOS functionality. Graphical VOS comes with a number of RLLs implemented as Dynamic Link Libraries (DLLs) that let you load executable code modules on demand to be linked at runtime. RLL functions can be called from a VOS application like other VOS functions.

For example, the Sockets RLL lets you send and receive messages to VOS tasks running on remote PCs that are connected via an IP network.

Debugger

The Debugger is a source-code level graphical debugger that lets you test and troubleshoot VOS programs through the development phase and after installation and deployment. You can use this tool to debug many programs in a single session and to check program variables and the current state of your application without interrupting the calls in progress.

To help you become more familiar with the Debugger, this guide includes a tutorial in which you create an application in Graphical VOS and then use the Debugger to debug it. For more information, see the Graphical VOS online help.

Note: The Debugger is not for use with CT applications developed in the CallSuite environment.

CallSuite

CallSuite is an alternative set of tools that you can use to add telephony features to legacy applications or to create new applications from scratch.

CallSuite is made up of Topaz, SimPhone, a collection of ActiveX components, and any Windows development environment that supports ActiveX components. The most commonly used programming language is Visual Basic, and others include Visual C++, Delphi, FoxPro, and PowerBuilder.

For details about SimPhone, see the section titled, “Additional Tools and Utilities” in this chapter.

CallSuite Components

The CallSuite ActiveX components add the following capabilities to your CT applications:

- VoiceBocx component —Handles call control, playing and recording of sound files and tones, and getting DTMF (Dual Tone Multi-Frequency) digits from callers
- FaxBocx component—Lets you add fax support to your applications
- ConfBocx component—Adds conferencing and call center switching capabilities to your applications

- ChatterBocx component—Controls text-to-speech features in your applications
- MatchBocx component—Enables your application to recognize speech
- CallSuite Wizard—Lets you automatically generate applications

Additional Tools and Utilities

Various additional tools, utilities, and sample applications are automatically included in the CT ADE installation and are available from the Start menu following installation. Depending on your purchase, the hardware key may also be included in your CT ADE package.

SimPhone

SimPhone is a software phone line simulator that works with either CallSuite or Graphical VOS and Topaz to simulate a telephony board and phone line. The simulation is accomplished using the screen, keyboard, mouse, and sound card installed in your PC. SimPhone can be used whether or not a telephony board is installed.

Sample Applications

Use the Start menu to access the sample applications for viewing or to modify them for use with your own applications.

CallSuite Samples

CallSuite has sample applications written in Visual Basic, Visual C++, and Delphi. The Visual Basic samples include applications that allow callers to do their banking by phone, send and receive faxes, and monitor both inbound and outbound calls. Additional applications use CT Media and NetHub Plus communication technologies. There are also several Voice Recognition and Text-To-Speech applications that utilize third-party software. Both the Visual C++ and the Delphi samples provide a multithreaded inbound call application.

Graphical VOS Samples

Graphical VOS has sample applications available in both the VOS language and FlowCharter. The Script samples consist of applications written using the VOS language. The FlowCharter samples were written using FlowCharter, which implements a user interface to VOS and provides a convenient way to use graphics and dialogs to build your CT applications.

The Script samples include numerous applications that let callers access movie title and showtime information, use an ActiveX control, perform database operations, and monitor inbound and outbound calls. Additional samples incorporate technologies such as Text-To-Speech, NetHub Plus communication, and Voice Recognition using third-party software.

The FlowCharter samples include various applications that let callers bank by phone, use a prepaid calling card, set up a conference call, send and receive faxes, monitor inbound and outbound calls, and retrieve voicemail. There are also two sample applications that use CT Media.

For information about CT Media, Text-To-Speech, Voice Recognition, and multithreaded applications, refer to the Graphical VOS or CallSuite online help.

Media Toolbox

Media Toolbox is a family of ActiveX controls that further help you modify your CT applications. It contains the following controls:

- Sound File Converter ActiveX and DLL for converting between Vox and Wave file formats
- IPFile ActiveX for creating and editing Indexed Prompt Files (also known as VAP, macro or VBase/40 files)
- WaveDevice ActiveX for querying the capabilities of your installed Wave devices
- WaveFile ActiveX for querying the format of a Wave file

There are also sample applications that you can use with your own sound files.

The Media ToolBocx executable is located in the following default installation directory:

```
C:\Program Files\Parity Software\Telephony Toolbox\Media
Toolbox
```

The samples and online help are located in:

```
C:\Program Files\Parity Software\Media ToolBocx
```

Audiotst

Audiotst is an application that determines the number of voice boards present on your system and then plays a set of Wave files in various formats to each board to find out if those formats are supported. If they aren’t, it reports an error message. You can use Audiotst with both Graphical VOS and CallSuite.

You can find the Audiotst executable and ReadMe file in the following default installation directory:

```
C:\Program Files\Parity Software\Common\Bin
```

WaveTest

WaveTest is a command-line utility that allows you to check your system for installed wave devices and to find out what their capabilities are. You can use WaveTest with both Graphical VOS and CallSuite.

You can find the WaveText executable and online help file in the following default installation directory:

```
C:\Program Files\Parity Software\Common\Bin
```

Hardware key

The hardware key, also called a dongle, must be attached to one of the parallel ports or USB on your PC in order for CT programs to function correctly. It has a standard parallel port connector and allows normal operation of printers and other devices that can be attached to the port. If you are running in evaluation mode the key is not required.

Caution:

We recommend removing the hardware key from the parallel port when not needed, especially when using bi-directional software such as At Work printers, LapLink and other file transfer utilities. Occasionally, some types of hardware devices can be damaged by use of bi-directional software or by static electricity.

Inside Topaz 2

This chapter describe the features of Topaz and how it works. It contains the following sections:

- Topaz Overview
- Topaz Profile
- Topaz.ini file
- Languages in Topaz

Topaz Overview

The Topaz portion of CT ADE is a set of software components that handles the CT technology functions in your applications. The main components of Topaz are the Scanner, the Profile, and the Topaz.ini file. Topaz also includes a collection of languages that you can use with your applications.

The Topaz Scanner is a utility that you run to gather information about speech engines and telephony boards installed on your system. It then builds a database called the Topaz Profile in which to store this information. Finally, the Topaz.ini file is used to specify which configuration parameters Topaz reads each time it is invoked on your system.

Topaz also stores a set of languages and administers information about them for use with Phrase functions. For example, if you want to speak a phrase in Italian, Topaz tracks where the Italian language prompts are located and uses the pre-defined rules included in the Italian directory to use the prompts correctly.

Figure 2-1 Flow of information through Topaz components

Topaz mediates between programming commands such as playing a file and the underlying hardware and API combination. While the various components of Topaz are referenced while your application is running, the object-oriented architecture of Topaz is structured so that only Topaz.dll operates at runtime and the performance of your applications is not compromised. The telephony boards and speech engines as well as their configuration and API combinations can be separated into groups called Technologies. Topaz gathers this Technology information from the Topaz Profile, translates these into usable Resources, and then uses them to carry out the CT commands you type.

For example, if you invoke the VOS function TrunkAnswerCall, the Topaz engine determines which trunk interface is being used, whether it is legal to make this call, and chooses which API function to use, such as dx_sethook (R4 analog), dt_setsig (R4 T-1), cc_Answer (R4 PRI),CTscr_AnswerCall (S.100 / CT Media), and so on.

Technologies are named using the convention <API><Resource Type> For example:

- R4DxMedia is a media resource based on Dialogic VOX and R4 dx_ API.
- R4AgTrunk is an Analog trunk interface based on Dialogic LSI and dx_/ag_ API. LSI interfaces are found on boards such as Proline/2V, Dialog/4, D/41ESC, VFX/40 and D/160SC-LS.
- S100Media is a media resource based on CT Media server and S.100 API.

There are many Technologies but each falls into one of the six Resource types.

Topaz Technology	Description	Resource Type
XXXTrunk	Responsible for all call control, including dialing out, accepting an incoming call, and hanging up when a call is finished	Trunk
XXXMedia	Controls all playing and recording of sound files and tones, as well as getting DTMF digits from callers	Media
XXXFax	Controls the transmission and processing of fax data	Fax
XXXVR	Controls voice recognition	VR
XXXTTS	Controls text-to-speech	TTS
XXXConf	Controls individual conferences	Conference

For a complete list of the Technologies currently available, refer to the Graphical VOS or CallSuite online help.

Topaz Resources

A Resource is a component of a call processing system. In most cases, a single Resource processes a single stream of audio corresponding to the sound on a telephone line or channel on a digital trunk. Technologies get interpreted into Resources by Topaz.

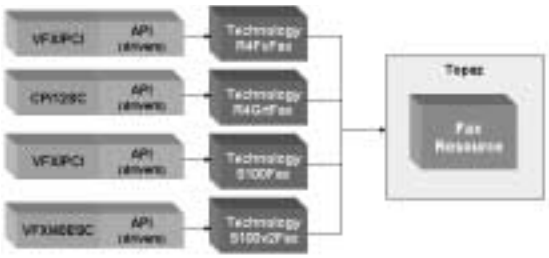


Figure 2-2 Topaz gathers Hardware and API configuration information and translates it into a Resource

Resources are identified when you run the Topaz Scanner and information about them gets stored in the Topaz Profile. Resources are managed by functions in Graphical VOS and methods or functions in CallSuite. Each Resource produces and/or consumes a single stream of audio:

Resource Type	Graphical VOS Function	CallSuite Component Method
Trunk	TrunkXXX functions	VoiceBocx methods
Media	MediaXXX functions	VoiceBocx methods
Fax	FaxXXX functions	FaxBocx methods
VR	ConfXXX functions	ConfBocx methods
TTS	TtsXXX functions	ChatterBocx methods
Conference	VrXXX functions	MatchBocx methods

When VOS or CallSuite and Topaz are started, each Resource is assigned a Resource index number, starting from 0 to the total number of that type of Resource on your system minus one. Generally, you won't have to worry about Resource index numbers, since Graphical VOS and CallSuite reserve and route Resources as your application needs them, but if you want to control how your applications use Resources, you can specify them by their index number. Index numbers start at 0 for each Resource type, and are numbered independent of other Resource types, so a VOS task could easily be using Trunk Resource 1 and Media Resource 4.

Individual Resources of a given Technology type are identified with Resource Index Numbers. So, for example, if your system had a D/240SC-T1, a D/41ESC, and a VFX/40ESC telephony board, you would have the following Topaz Resources:

D/240SC-T1:	24 Trunk Resources	24 Media Resources	
D/41ESC:	4 Trunk Resources	4 Media Resources	
VFX/40ESC:	4 Trunk Resources	4 Media Resources	4 Fax Resources
Totals:	**32 Trunk Resources**	**32 Media Resources**	**4 Fax Resources**

This system's Trunk Resource index numbers range from 0 to 31, and the Media Resource index numbers also range from 0 to 31. The Fax Resource index numbers range from 0 to 3.

To find out how to find the number of Resources available on your system, see either the Graphical VOS or CallSuite online help.

Trunk Resources

Trunk Resources are responsible for all call control. Call control includes dialing out, accepting an incoming call, and hanging up when a call is finished. Because a Trunk Resource processes a single stream of audio, each of the following is considered a single Trunk Resource:

- The trunk interface for a single analog line
- One T-1 or E-1 time-slot
- An MSI station
- SimPhone's simulated trunk line (always Index number 0)

For example, if you invoke the VOS function TrunkAnswerCall, the Topaz layer determines which trunk interface is being used, whether it is legal to make this call (has an incoming call been signaled?), and then which API function to use: *dx_sethook* (R4 analog), *dt_setsig* (R4 T-1), *cc_Answer* (R4 PRI), *CTscr_AnswerCall* (S.100 / CT Media), and so on.

Media Resources

Media Resources control the playing and recording of sound files and tones, as well as getting DTMF (Dual Tone Multi-Frequency) digits from callers. For example, a voice recorder that records the caller's response would be considered a Media Resource.

Fax Resources

Fax Resources control the transmission and processing of fax data. A single fax channel on a Dialogic VFX board or on a GammaLink CP board is considered to be one Fax Resource.

It's important to remember that Fax Resources can only send and receive fax data. All other functions that are required to answer and make telephone calls are performed by Trunk and Media Resources.

Voice Recognition Resources

Voice Recognition (VR) Resources translate a caller's spoken input into text strings. One VR Resource can perform a recognition on a single stream of audio data from one Trunk or Conference Resource.

Note: Topaz creates one Voice Recognition Resource for each Media Resource on your system. Depending on your speech engine's limits, you may not be able to use all of these VR Resources simultaneously. See the topic titled, Voice Recognition Resource States in either the Graphical VOS or CallSuite online help for more information.

Topaz VR Technologies also include those of several third-parties, for more information about third-party VR Technologies, see either the Graphical VOS or CallSuite online help.

Text-To-Speech Resources

Text-To-Speech (TTS) Resources translate text strings into spoken output. One TTS Resource can speak with a single stream of audio data to one Trunk or Conference Resource.

For example, you might create an application in which callers can call in remotely to access their e-mail and request that messages are read to them over the phone.

Note: Topaz creates one Text-To-Speech Resource for each Media Resource on your system. Depending on your speech engine's limits, you may not be able to use all of these TTS Resources simultaneously. See the topic titled Text-To-Speech Resource States in either the Graphical VOS or CallSuite online help for more information.

Topaz TTS Technologies also include those of several third-parties, for more information about third-party TTS Technologies, see either the Graphical VOS or CallSuite online help.

Conferencing Resources

Conference Resources control individual conferences. Each conference, regardless of how many parties it contains, is controlled by a single Conference Resource.

Note: Sometimes the documentation included with your telephony boards uses the term *conference resource* to refer to the number of parties an MSI or DCB board can place into conferences. In Topaz, a *Conference Resource* controls the logical part of a conferencing board that manages a single conference.

Resource States

For every kind of Resource (Trunk, Media, Fax, TTS, VR, or Conference), Topaz pre-defines a number of states. For example, a Trunk Resource can be in a Ringing or Disconnecting state, and a Media Resource can be in a Playing or Recording state.

The state of a Resource can change in two ways: unsolicited (as a result of an external event), or as a result of a Topaz function/method call. An example of an unsolicited state change is a Trunk Resource that is currently Idle transitioning to a Ringing state when an incoming call is signaled. An example of a state change due to a function call is a Media Resource that is currently Idle transitioning to a Playing state in response to a Play Command.

Functions that can change the state of a Topaz Resource are known as Commands. For example, the VOS function MediaPlayFile invokes the Topaz Play Command.

Each Resource can have a variety of states such as Busy, Idle, Recording, and so on. Topaz defines the following state parameters for each Resource type:

- The set of possible states
- The set of Commands that can change the state
- The state(s) in which a given Command may be issued
- The state to which a given Command can take a Resource (always one pre-defined state)
- The beginning and ending states of unsolicited state changes

Collectively this information is called a state diagram. The following graphic represents a simplified sub-set of the Topaz Media Resource state diagram.

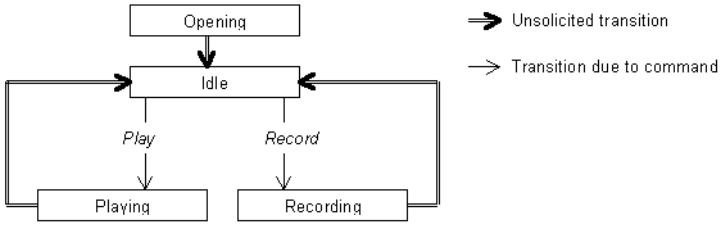


Figure 2-3 Topaz Media Resource state diagram

The Topaz Media Resource state diagram has four states: Opening, Idle, Playing, and Recording, and two Commands: Play and Record. The diagram illustrates that it is illegal to issue the Play or Record Command unless the Resource is in the Idle state. If the Play Command is successful, the Resource is always in the Playing state. At some point in the future following a Play Command, the Playing state always transitions back to the Idle state (when the end of the file is reached).

Every Topaz Resource must conform to the rules specified in the state diagram for its type. However, a given Resource may implement only a sub-set of the diagram. For example, a loop-start analog trunk (a Plain Old Telephone Service line like you probably have at home) has no signal corresponding to the “out of service”, “out of sync” or “D channel down” conditions found on digital trunks. Topaz defines an unsolicited transition from Idle to OutOfService for Trunk Resources. Digital Trunk Resources may experience this transition, but an analog Trunk Resource will never do it. Just because some Topaz Resources support certain state transitions and Commands does not mean that all Topaz Resources of that type support them.

A graphical representation of a simple state diagram like the one above is easy to grasp. However, for more complex state diagrams this representation can become unwieldy and confusing. Usually, Topaz state diagrams are presented in table form. For example, the information in the state diagram can be equally well represented by the following table.

Start State	Command	End State
Opening	—>	Idle
Idle	Play	Playing
Playing	—>	Idle
Idle	Record	Recording
Recording	—>	Idle

Unsolicited state transitions are indicated by → in place of a Command.

For more information about state diagrams for each of the Resource types, refer to either the Graphical VOS or CallSuite online help.

Topaz Profile

The Topaz Profile is a database that is organized into a tree of directories and files similar to the Windows registry structure, and stores this information:

- Details of all installed hardware and API combinations (Technologies)
- User-supplied hardware configuration information
- User-configurable options, such as the default language for speaking values (examples are English and Spanish)

The Topaz Profile files are located in the following default installation directory:

```
C:\Program Files\Parity Software\Common\Tpaz\Profile
```

Unfortunately, the database is not directly readable. In order to view and modify the information in the Topaz Profile after running the Topaz Scanner utility, type TopazProfile.exe -L from a command prompt. This populates theTopazProfile.txt file which mirrors the database information and is organized in a format that you can read and understand. Each entry in the generated file represents a Profile ID.

You can also view the TopazProfile.log file to find out what resources were found on your system during a scan as well as any additional files included in the scan.

Both the TopazProfile.log and the TopazProfile.txt files are located in the following default installation directory:

```
C:/Program Files/Parity Software/
```

No dynamically changing information, such as the current state of a Resource, is stored in the Topaz Profile. When an application is running it treats the Profile as read-only so the Profile must be fully initialized before an application is started.

Topaz Profile IDs

Topaz Profile IDs have a name and a corresponding value that describe system details. Internally, the Topaz Profile does not store the character strings shown for value names. All value names are actually stored as integers. The set of integers that may be used for value names is pre-defined for a given Topaz release. Several Topaz utilities convert between the integer values used internally and the string names to display the Profile in a form that is convenient

for a human to read. This allows for more efficient lookup algorithms and hence faster access to the Topaz Profile when Topaz applications are running. The allowed integer values for value names are called Topaz Profile IDs.

You can obtain the current value of a Profile ID with the appropriate Get function. For example, in Graphical VOS, to find out if ANI (Caller ID) support is enabled, you would use the Trunk GetBool function with REGID_ANISupported (301):

```
Support = TrunkGetBool(301);
```

The Profile ID Tables topic lists the Profile IDs that are valid for each Technology.

The name of an entry is similar to a path name in a file system. All names begin at the root, which is designated by a back-slash character (\). For example, a Profile ID much used by Topaz code internally is:

```
\TechCount
```

The value of TechCount is the number of different Topaz Technologies installed in this PC. Values are one of three types: integer, string, or Boolean (True / False). A convenient shorthand which you may often see shows the value name and value like this:

```
\TechCount=4
```

Topaz RegIDs

RegIDs are values used internally by Topaz as well that access technology-level information. You can use them with the Get/Set functions to retrieve information about the system or to issue a Technology-specific command at runtime.

Some RegIDs describe system details and these are stored in the Topaz Profile—these RegIDs are called Profile IDs. You can get the current value of a Profile ID with the appropriate Get function/method. For example, to find out if ANI (Caller ID) support is enabled in CallSuite, you would use the GetTrunkBool method in CallSuite with REGID_ANISupported (301):

```
Support = VoiceBocx1.GetTrunkBool(301)
```

To find out which Profile IDs are valid for each Technology, see either the Graphical VOS or CallSuite online help.

TZP Files

TZP files are also called *Profile Include files* because they hold additional information that the Topaz Profile uses to perform various telephony and language functions. CT ADE comes with four TZP files that you can view and

modify using a text editor like Notepad. You can also create your own TZP files, but you must save them with a .tzp extension and store them in the TZP directory.

The TZP files are located in the following default installation directory:

```
C:/Program Files/Parity Software/Common/Topaz/TZP
```

To add the information contained in a TZP file to the Topaz Profile so that Topaz can use it, you need to make sure it gets included during scan time. To do this, you use the Topaz.ini file. For more information about using the Topaz.ini file, see the section titled, "Topaz.ini File" in this chapter.

IPFs.tzp File

The IPFs.tzp file designates a number to each of the fourteen languages that come with the CT ADE installation and stores a set of parameters for each.

Figure 2-4 IPFs.tzp file displayed in Notepad

The following Indexed Prompt File parameters are also called Profile IDs. For more information about Profile IDs, see the section titled, "Topaz RegIDs" in this chapter.

- IPFName—Used by functions or vocabulary entries to specify which IPF file contains a desired prompt
- IPFFile—Specifies the name and path to the IPF File
- AudioFormatCoding—Specifies whether the file was recorded with A-law or mu-law companding. Valid values for AudioFormatCoding are Alaw or muLaw.
- AudioFormatSamplesPerSecond—Specifies the sampling rate, measured in samples per second (Hz). VOX files use 6 kHz or 8 kHz, so valid values for AudioFormatSamplesPerSecond would be 6000 or 8000.
- AudioFormatBitsPerSecond—Specifies the amplitude (loudness) of the sound at the instant a measurement was taken represented by a binary number. Sample sizes are generally 4-, 8- or 16-bit. Valid values for AudioFormatBitsPerSample are 4, 8, or 16.

- AudioFormatChannelCount—Specifies whether a file is recorded in stereo, which has a channel count of 2, or mono, which has a channel count of 1

For more information about this file, see "IPF" in this chapter.

Tip:

To find out whether your modified TZP file/s got included in the scan, run the TopazProfile .exe then open the TopazProfile.log file and check the include files listed under the heading:
`These files are included in Topaz.ini [ProfileIncludeFiles]`

Running the Topaz Scanner

When you run the Topaz Scanner Utility (TopazProfile.exe), it creates a database called the Topaz Profile

Also called the Resource Scanner, the Topaz.exe utility scans your system hardware and collects information on any installed telephony boards as well as their configuration details. The Scanner then builds a database called the Topaz Profile and places the collected information into it.

If you want, you can merge additional information that isn't collected during a scan into the Topaz Profile or delete information from it by using *TZP files*.

The Topaz Scanner automatically runs upon installation, but you must rerun the Scanner each time you make a change to your current configuration.

To run the Scanner:

1. From a command prompt, locate the following default installation directory:

   ```
   C:/Program Files/Parity Software/Common/Topaz/Bin
   ```

2. From the Bin directory, type:

   ```
   TopazProfile.exe -S -L
   ```

   Topaz extracts all the available configuration information from your telephony boards and categorizes these into one of the six Resource API types. Next you need to update the TZP files with additional information that Topaz cannot automatically determine.

3. Open and modify the appropriate TZP file/s located in the following directory:

   ```
   C:/Program Files/Parity Software/Common/Topaz/TZP
   ```

   For example, for an E-1/R2 trunk, you must specify which country-dependent parameters should be used for the R2 protocol. For an analog trunk, you must specify whether caller ID is available. For a T-1 trunk, you must specify if ANI and/or DNIS is available, whether ANI/DNIS

(Automatic Number Identification /Dialed Number Identification Service) is provided through DTMF or MF digits and how many digits are sent, and so on.

4. If you've updated one or more TZP files with additional modifications, you need to run the TopazProfile.exe utility again, this time using the -C option instead of the -S option to merge the modified TZP file/s into the Topaz Profile. If you haven't modified a TZP file, skip this step.

Topaz.ini File

The Topaz.ini file is used to specify various configuration parameters area that are read by Topaz each time it is invoked on your system. If you change any entries in Topaz.ini, you must be sure that all instances of VOS/CallSuite applications and any CT ADE utilities such as TopazProfile.exe or Phraser.exe (which use Topaz) are closed before the changes can take effect.

You can edit the Topaz.ini file with a text editor such as Notepad, but the file must reside in the same directory where Windows is installed. Typically this is in the C:\WINNT directory.

Figure 2-5 Topaz.ini file

The Topaz.ini file is populated with several sections by default (see the Topaz.ini graphic) but you can also add additional sections for Topaz to use.

Topaz.ini file sections:

- [NFAS]—Lists NFAS boards that don't use a D channel
- [Profile]—Topaz Profile startup information
- [ProfileDependencies]—Windows NT/2000 services that must be started before the TopazProfile.exe utility starts
- [ProfileExcludeTechnologies]—Telephony technologies to exclude from the Topaz Profile scan
- [ProfileIncludeFiles]—Include files to be merged with the Topaz Profile during AutoMerge

- [S100Groups]—S100 groups and their corresponding Topaz Resources that are used by your application
- [S100Defaults]—Defaults for keys in the S100Groups section
- [SimMediaScanner]—IDs of the Wave In and Wave Out devices you want SimMedia to use
- [SpeechPearl]—Philips SpeechPearl configuration information
- [SpeechWorks]—Defines DialogModules, services, and vocabularies for the SWVR Technology
- [WaveScanner]—IDs of the Wave In and Wave Out devices you want WaveMedia to use

Languages in Topaz

The CT ADE installation comes with fourteen different languages that Topaz uses to translate audible information (Text-To-Speech capability) to a caller. A language is a set of recorded words and phrases in a particular language (and possibly gender) along with the supporting files needed for Topaz to find and properly use the prompts. Typical phrases generated by call processing applications might be, *"You have twenty-three new messages,"* or, "*Your current balance is nineteen dollars and one cent."* Phrases consist of pre-recorded sentences or parts of sentences like "You have" and variable information like "twenty-three."

Variable information is constructed by stringing together pre-recorded prompts. For example, the money amount $19.01 would be generated by VOS from the following five prompts:

"nineteen" "dollars" "and" "one" "cent"

By default, your application is able to generate variable information for numbers, money amounts, dates, time, and ordinals in a number of languages using prompts from one or more files that are provided with the default installation.

You can also create your own language if you want. For instructions on how to do this, refer to the Graphical VOS or CallSuite online help.

Language Files

All the information about a language is divided into four main groups:

- IPF (Indexed Prompt File)
- Language Profile

- TZP files
- Optional text files

IPF

The Indexed Prompt File (IPF) contains VOX files, also known as prompts, which are recorded words or phrases. Compiled with an .ipf extension, each IPF begins with a header that includes an indexed listing all of the included prompts. For more information about this file, see “IPFs.tzp File” in this chapter.

Language Profile

The Language Profile file contains attributes for a language, called vocabularies, as well as rules that govern the playing of those prompts. This set of rules is called a grammar. Stored as an .LNG file, Topaz uses this file to determine which prompts to play and when.

Both the IPF and the Language Profile file are stored in the language default installation directory:

```
C:\Program Files\Parity
Software\Common\Topaz\Language\<Language_1>
```

TZP Files

The Topaz Profile Include files, also called TZP files, are a pair of text files containing entries that are added to the Topaz Profile via the Topaz.ini file. (The Topaz.ini file contains paths to all the available languages and gets merged into the Topaz database at startup time.)

There are two types of TZP files: an IPF include file, and a Language include file. Together they provide Topaz with a “map” to the locations of the IPF and Language Profile for a particular language:

- IPF include file – A TZP file that maps to the location of your language’s Indexed Prompt File.
- Language include file – A TZP file that maps to the location of your language’s Language Profile

Languages.tzp File

The Languages.tzp file designates the same number for each language that the IPF.tzp file does and stores each language’s directory location so Topaz can find it on your system. In the following graphic, English is designated the number 0, and its file location is displayed on the next line.

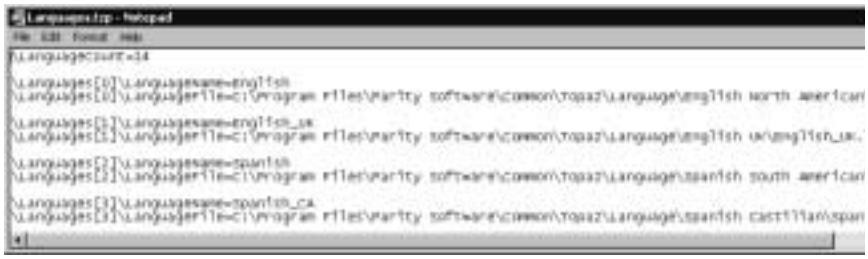


Figure 2-6 Languages.tzp file displayed in Notepad

The TZP files are located in the following default installation directory:

```
C:\Program Files\Parity Software\Common\Topaz\TZP
```

Text Files

Optionally, a language can also have two text files that are used to test words and phrases:

- Language.TST file – Text file the Phraser.exe uses to test prompts
- Language.TXT file - Text file listing the names of all your prompts

Both of these files are stored in the language directory along with the IPF and Language Profile file.

Phrase Element Parameters

In Topaz, recorded words are known as elements, which can be strung together to form phrases. For example the phrase *"three hundred twenty-three"* can be broken up into the elements, "three," "hundred," and "twenty-three". There are four parameters required for every phrase element: Value, Type, Gender, and Language. You set these parameters in your application and then Topaz uses the parameters to play phrases.

Value Parameter

The Value parameter is passed to Topaz and used in conjunction with the Type parameter to provide the element's content. For example, a value of 330 may produce a spoken phrase of *"three hundred and thirty"* or *"three-thirty, p.m."* depending on the Type parameter used. Topaz determines the Value parameter from the parameters in the Graphical VOS Play function/CallSuite Play method you use.

Type Parameter

The Type parameter tells Topaz how to interpret the Value. Some of the built-in Types include Numbers, Ordinals, Dates, Times, and Money. To see how the Type parameter can affect the way element is spoken, let's look at a couple of examples that have the same Value, but different Types.

Value	Type	Spoken
56	Numbers	fifty-six
56	Ordinals	fifty-sixth
123456	Numbers	one hundred twenty-three thousand, four hundred fifty-six
123456	Times	twelve thirty-four and fifty-six seconds p.m.
20020301	Numbers	twenty million, twenty thousand, three hundred one
20020301	Dates	March first, two thousand two

The Play method/function you use determines the Topaz phrase Type. In Graphical VOS for example, if you call the PlayOrdinal function, the Type is set to Ordinals. If you call the PlayValue function, you can manually set the Type with the ValueType parameter.

In addition to the Types that come with Topaz, you can create your own to accompany an existing language or as part of a new language. For more information about Type parameters, refer to either the Graphical VOS or CallSuite online help.

Gender Parameter

Gender refers to the gender to use for the element, that is, the gender of the word itself, not the gender of the voice speaking the prompt. For example, the French for “one book” is “un livre,” which is masculine, so if you were constructing a phrase that gave a number of books, you would set the Gender parameter to masculine. Topaz uses the value of the Gender to determine the second digit in the Grammar label to jump to.

Gender	Value
neuter	0
masculine	1
feminine	2

In CallSuite for example, Topaz determines the Gender by reading the VoiceBocx Gender property, so to speak the word “seven” before a feminine noun, you would use:

```
VoiceBocx1.Gender = 2
Call VoiceBocx1.PlayInteger(7)
```

In English, the gender will always be neuter since English adjectives aren't gendered.

Gender is an integer value which may be used to modify how a value is spoken, or may be ignored, depending on the grammar file. For example, in English grammars, it is usually ignored since English has no concept of gender. In French, the Gender property would determine whether PlayInteger(1) speaks “un” (Gender=1, masculine) or “une” (Gender=2, feminine). German has three genders, and so on.

Even in English, however, the Gender can provide a “hook” for special purposes with user-defined data types. For example, should a time be spoken using 12- or 24-hour clock?

Language Parameter

The Language parameter tells Topaz which Language Profile and IPF to use when processing a phrase element. The Language parameter must match the name of a language name set in the language include file. The Language is a string that gets matched against a language registered in the Topaz Profile. If the Language is an empty string, the default will be the first language specified in the Topaz profile.

Using SimPhone 3

This chapter describes how to use SimPhone with Graphical VOS and CallSuite and contains the following sections:

- Overview
- Starting SimPhone

Overview


SimPhone icon

SimPhone is a phone simulator application that performs telephone functions for your CT applications and does not require telephony hardware to be present in your PC. This can be helpful, for example, if you are on a business trip and the only computer available is a laptop. SimPhone works with both CallSuite and Graphical VOS applications, including Graphical VOS applications built with FlowCharter or running in the Debugger. You can also demonstrate a finished application to a client using SimPhone and a standard PC with a sound card.

SimPhone supports a variety of Vox and Wave formats. It can also play and record sound files in real-time and with high quality using your SoundBlaster™ or other Wave-compatible sound card. For more information about playing and recording files, see either the Graphical VOS or CallSuite online help.

Starting SimPhone

To launch the SimPhone application, select SimPhone from the Start menu. In Graphical VOS, you can also start SimPhone from the main menu by choosing Run | Launch SimPhone. SimPhone always simulates Trunk Resource 0 and Media Resource 0.

For more information on Trunk Resources or Media Resources, refer to the Graphical VOS or the CallSuite online help.

Note: If you start SimPhone when an application is already running, VOS/your CallSuite application will continue to use the telephony card.

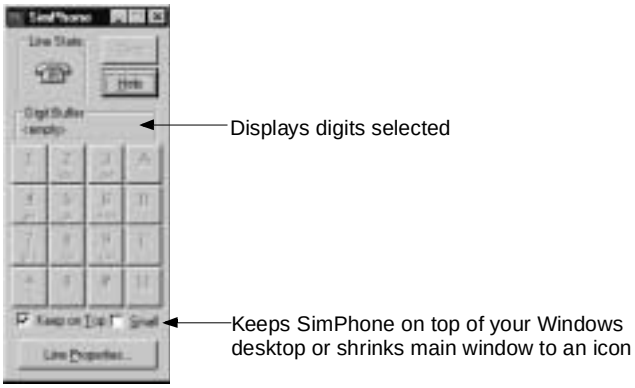


Figure 3-1 SimPhone main window at startup

After you've started SimPhone, it's time to start your application.

Incoming Calls

Most applications wait for an incoming telephone call. You can send a call to your application by clicking the Ring button. This triggers an incoming call event to SimPhone just like a ring on a telephone line would. Your application usually responds by answering the call. SimPhone displays the current state of the simulated telephony channel's hook switch by the Line State icon: when the line is off hook, the off-hook icon is shown, and when the line is on hook, the on-hook icon is shown.

Line State Icon	Description
	Simulated phone line is on hook. If the icon is grey, SimPhone is not being used
	Simulated phone line is off hook

When the line is off hook, the Line State icon shows a telephone with the handset picked up (off hook). The *close* system button and menu option are disabled (they appear grey instead of black), the Ring button becomes the Hangup button, and each DTMF (Dual Tone Multi-Frequency) Digit button is enabled. Clicking the Hangup button simulates a caller hang-up.

Pressing digit buttons has the effect of adding the digits to the digit buffer on the simulated channel and the current contents of the buffer are shown in the Digit Buffer display. The buffer will be reduced or emptied when the appropriate VOS functions/CallSuite methods are invoked.

You cannot exit SimPhone while an application is in progress. This is indicated by the greying of the close system button and menu option. This button will be enabled again when you stop your application.

Phone Line Properties

SimPhone lets you set the ANI (Automatic Number Identification), DNIS (Dialed Number Identification Service), and caller name that are reported to your application when you make an inbound call. Before placing a call to your application, you must set the phone line properties by clicking the Line Properties button at the bottom of the SimPhone main window, and then completing the appropriate fields.

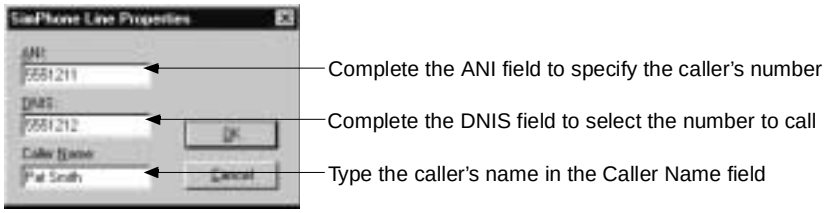


Figure 3-2 SimPhone Line Properties dialog

Click OK, and the new settings take effect with the next call to your application.

Outbound Calls

SimPhone can also receive outbound calls from VOS or CallSuite applications.

When your application places an outbound call that requires Call Progress Analysis (for example, if you use the Dial with Analysis cell in a flow chart or if you use the TrunkMakeCall function in the VOS language), SimPhone displays a dialog that allows you to specify the result of the call:

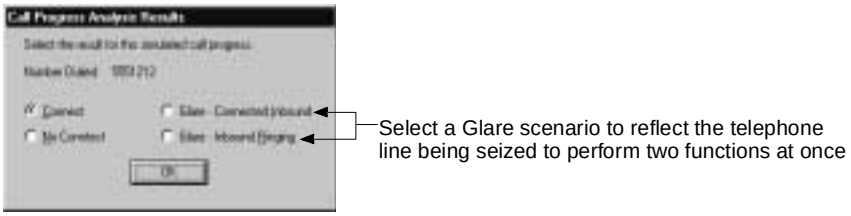


Figure 3-3 Call Progress Analysis Results dialog

Select one radio button to indicate what call scenario you want Call Progress Analysis to return. Click OK, and SimPhone simulates the result you chose.

For more information about SimPhone, refer to the SimPhone online help.

Getting Started with CallSuite 4

This chapter introduces CallSuite and contains the following sections:

- Overview
- Tutorial 1: Generating a Starter Application in CallSuite
- Tutorial 2: Modifying Your CallSuite Starter Application

Overview

The CallSuite portion of CT ADE provides a specialized platform that lets you reduce the repetitive tasks associated with a traditional telephony application development, and enables you to stay focused on the essential and profitable aspects of developing your application.

Made up of a collection of ActiveX components that includes an application wizard as well as a wizard utility, CallSuite makes it easy for you to add CT functions to your existing systems or to create applications from scratch. The ActiveX components can be used in any Windows development environment that supports ActiveX components. The CallSuite Wizard utility provides a GUI environment where you can create new CT functions, and depending on how your application was developed, edit existing ones.

You develop applications in CallSuite one of two ways: by automatically generating an application using CallSuite Application Wizard, or by writing your application directly in the programming language of your choice and adding ActiveX components to it.

If you use the CallSuite Application Wizard to automatically generate an application for you, you can view all of the CT functions contained in your application from the CallSuite Wizard. From this main window you also can modify or remove functions as well as add new ones.

Alternatively, if you initially created your application using a language other than the Callsuite Application Wizard then you cannot use the CallSuite Wizard to add new CT functions to the project. You must add and edit functions directly in the code.

CallSuite Components

The CallSuite set of ActiveX components can integrate directly into visual programming environments such as Visual Basic, Visual C++, or Delphi, and provide CT-specific development functionality.

CallSuite components are invisible at runtime and do not have an interface, but you can easily create one using the conventional visual controls such as buttons, scroll bars, etc., so that your system administrator can more easily facilitate the application during runtime.

VoiceBocx

VoiceBocx gives you control of trunk and media hardware through Topaz Trunk Resources and Media Resources by handling call control, the playing and recording of sound files and tones, and getting DTMF digits from callers.

Trunk Resources are responsible for all call control, which includes dialing out, accepting an incoming call, and hanging up when a call is finished. Media Resources control all playing and recording of sound files and tones, as well as getting DTMF digits from callers.

FaxBocx

FaxBocx adds fax support to your applications by giving you control of fax hardware through Topaz Fax Resources. FaxBocx works closely with VoiceBocx and the other CallSuite controls. In your programs, you use VoiceBocx first to control call processing functions such as answering or placing phone calls. Next, you use FaxBocx to process the fax information. Finally, you use VoiceBocx again to finish processing the call.

ConfBocx

ConfBocx lets you manage conferencing capabilities in your applications. You also control conferencing hardware through Topaz Conference Resources. ConfBocx works closely with VoiceBocx and the other CallSuite controls. In your programs, you use VoiceBocx first to control call processing, such as answering or placing phone calls, playing menus to callers, and getting the digits they type in response. Next, you use ConfBocx to connect to a conference, and then to disconnect from the conference. Finally, you use VoiceBocx again to finish processing the call.

ChatterBocx

Chatterbox lets you control text-to-speech features in your applications through Topaz Text-To-Speech Resources. ChatterBocx works closely with VoiceBocx and the other CallSuite components. In your programs, you use VoiceBocx first to control call processing, such as answering or placing phone calls, playing menus to callers, and getting the digits they type in response. Next, you use ChatterBocx to speak text to the caller. Finally, you use VoiceBocx again to finish processing the call.

MatchBocx

MatchBocx lets you add speech recognition capabilities to your application through Topaz Voice Recognition Resources. MatchBocx works closely with the VoiceBocx control. In your programs, you use VoiceBocx first to control call processing, such as answering or placing phone calls, playing menus to callers, and getting the digits they type in response. Next, you use MatchBocx to recognize speech from the caller. Finally, you use VoiceBocx again to finish processing the call.

CallSuite Wizard

CallSuite Wizard is a visual application generator that writes, edits, and tests telephony application source code. The CallSuite Application Wizard is a series of dialogs that you use to create starter applications in one of several popular visual tools such as Visual Basic, Visual C++, Delphi, and PowerBuilder. So in order to create an application using CallSuite you use: the programming language of your choice, the CallSuite Application Wizard (this is optional), and the CallSuite Wizard. You start by opening the programming language in which you plan to create your application; you can choose from Visual Basic, Visual C++, or Delphi.

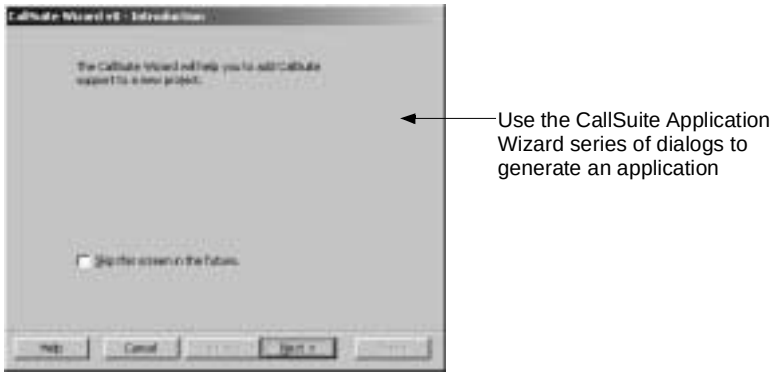


Figure 4-1 CallSuite Application Wizard—first dialog

Next, you launch the CallSuite Application Wizard from within the programming language. As you progress through the wizard dialogs, you specify how you want to set up your application and then the wizard generates a project containing your application. The telephony functions in applications are called Routines, which are series of source code statements you can invoke using a single name. Generated applications contain two sets of Routines: those that you chose in the Wizard, and those that are automatically generated (such as error message Routines and timeout Routines).

Finally, you use the CallSuite Wizard main window to change your application to suit your needs. For example, you can further define existing Routines, such as rerecording the default greeting or changing which dialed digits will generate an error message. You can also add new Routines to your application, such as a GetDigits Routine that captures the digits a caller dials.

Warning:

If you generate an application using CallSuite Application Wizard, you cannot edit the generated code directly in the program language because any edits you make manually are wiped out when you save your work in the CallSuite Wizard.

To view the CallSuite Wizard, choose Start | Programs | Parity Software | CallSuite | CallSuite Wizard.

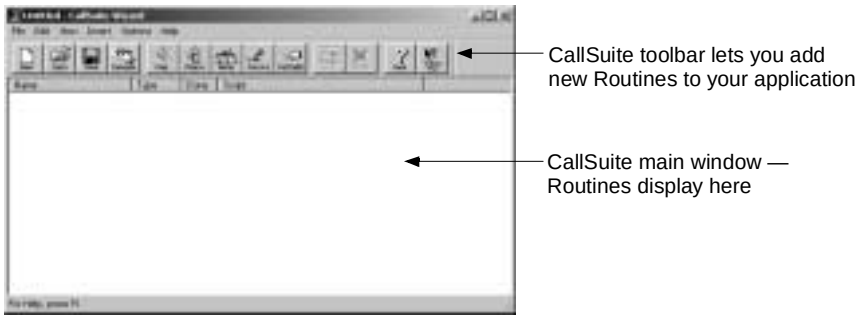


Figure 4-2 CallSuite Wizard—main window

For additional information, you can access the CallSuite Wizard help file from the CallSuite Wizard main menu.

This chapter includes two tutorials that use CallSuite Application Wizard and CallSuite Wizard. In Tutorial 1 you create an application in Visual Basic using CallSuite Application Wizard. In Tutorial 2 you make modifications to your application using the CallSuite Wizard.

Media Toolbox

Media Toolbox is a family of ActiveX components that you can use to further manipulate telephony sound files. After installing the utility you have access to a host of tools including a sound file converter component, a component that helps you create and edit Indexed Prompt Files (IPFs), a component that can query the capabilities of your installed Wave devices, and sample applications.

CallSuite Programming

As you develop and run applications using CallSuite, you use methods to send commands to Topaz, define properties to describe the attributes of the various objects you are using, and monitor events to find out what actions have occurred during development and runtime.

Methods

Methods are functions that implement most actions in your program. You send commands to Topaz requesting specific CT functions.

You can run CallSuite in synchronous or asynchronous mode. In synchronous mode, methods put the calling task or thread to sleep (this is called blocking) until the requested operation completes.

In contrast, when asynchronous mode is enabled, supported methods return immediately and indicate whether the operation started successfully while the operation continues in the background. Later, your application receives an event reporting that the operation has completed. Asynchronous mode is disabled by default. Many (but not all) CallSuite methods are affected by asynchronous mode; to find out which methods are affected, see the CallSuite online help.

Properties

A property is a data value similar to a variable and is generally used for attributes of an object that tend to remain fixed. In CallSuite for example, the ElapsedSecs property returns the number of seconds that elapsed during the last Play or Record. Properties are specified using the name of the object and the name of the property separated by a period. You set property values at runtime.

Events

Events are reports from the object to its client signifying that an action has occurred. Many events have corresponding methods that wait for a specified action to take place. For example, if an object displays a button on the screen, a typical event may report that the button was clicked. In CallSuite, you can process events by defining a function to be called when the event is triggered.

Tutorial 1: Generating a Starter Application in CallSuite

In this tutorial you use CallSuite Wizard and Visual Basic to create a simple outbound calling application that does the following:

- Makes an outbound call
- Plays a greeting when SimPhone answers the call, and gives this menu choice to the caller:

  - — Hear the company's stock price
  - — Contact Technical Support
- Performs the requested action or plays a message if an invalid option is chosen
- Lets the caller choose again until he or she hangs up or chooses to end the call
- Plays a goodbye message and hangs up

Before you Begin

Make sure you have CT ADE with CallSuite 8.0 (or higher), SimPhone, and Microsoft Visual Basic 6.0 installed on your system. If you have questions or want more information on CallSuite Application Wizard or the other wizard components, see the CallSuite Wizard online help. SimPhone is used in this tutorial to simulate a phone call to the voice card.

Step 1: Using CallSuite Application Wizard

In this section you use CallSuite Application Wizard to create a new application.

1. Start Visual Basic and close any open projects.
2. From the Add-Ins menu in Visual Basic, choose CallSuite Wizard v8.

   The CallSuite Wizard starts and displays the Introduction dialog.
3. Click Next to display the Step 1 dialog.
4. Select Outbound to create an outbound calling application. Then click Next to display the Step 2 dialog.
   a. In the Project Name field, type **VBSampleOutbound**.
   b. In the Project Path field, type the following path **C:\temp\VBSample**.
   c. Click Next to display the Step 3 dialog.

   Here we want to add a main menu in our application following the greeting. The starter application always begins with a greeting because most call processing applications begin with one such as:

   "Hello and thank you for calling..."

   —and follow with a main menu such as:

   "For this option, press 1, for this other option, press 2..."

   You can choose to add a main menu that has from two to nine options.

5. In the Step 3 dialog:
   a. Click Yes to add a menu to your application.
   b. Type 2 in the field below to specify two menu options.
   c. Click Next to display the Finished dialog.
6. Click Finish to create the project.

   CallSuite Wizard creates an outbound call processing application in the directory you specified in step 4. A message box tells you when CallSuite Wizard has created the Visual Basic project.
7. Click OK.

   Once the project is created, the Visual Basic Project Window shows the new form and module:

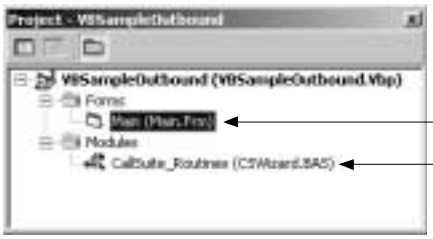


Figure 4-3 Project window in Visual Basic

Running the Application

Now you can run your new application.

1. Run the application from Visual Basic by choosing Run | Start.

   The Outbound with CPA dialog and the SimPhone main window appear.
2. From the Outbound Telephony Application dialog, click Dial.

   The Call Progress dialog displays. If you don't see this dialog, click anywhere on the SimPhone main window to bring it into focus on your desktop.
3. From the Call Progress dialog, click Connect, then click OK.

   The call is connected and a welcome message plays:

   "Hello, and thank you for calling Parity Software. Make your selection now, or press star to end this call."

Note that the Line State in the SimPhone main window, which is represented by a telephone, goes off hook when the call is answered.

4. From the SimPhone main window, press star to end the call.

   A goodbye message plays:

   "Goodbye, and thank you for calling."

   The application hangs up and the Line State in the SimPhone main window goes back on hook.

 *Caution:*

Dialing any digit other than the star key in step 4 results in an application error because a message hasn't been recorded for alternative choices yet.

---

5. From the Visual Basic main window, choose Run | End, and then click the close icon in the SimPhone main window.

Tutorial 2: Modifying Your CallSuite Starter Application

Now that you have created a project that contains a starter application, you can begin modifying existing routines and adding new ones. In this section you modify the Routines in your application and then test it.

 *Caution:*

You must use CallSuite Wizard to edit any Routine in an application generated using Callsuite Application Wizard. If you make changes directly to the code, they are erased the next time you open the project in CallSuite Wizard.

Before you Begin

In this tutorial you modify the application you created in Tutorial 1: Generating an Application in CallSuite. If you haven't completed the steps in Tutorial 1, go back and finish it before starting this tutorial. SimPhone is used in this tutorial to simulate a phone call to the voice card.

Step 1: Modifying the Main Menu Prompt Routine

In this section you modify the message in the Main Menu Routine.

1. Launch Visual Basic, and from the main window, open the VBSampleOutbound.Vbp project. Then choose Add-Ins | CallSuite Wizard v8.

   The Callsuite Wizard displays.

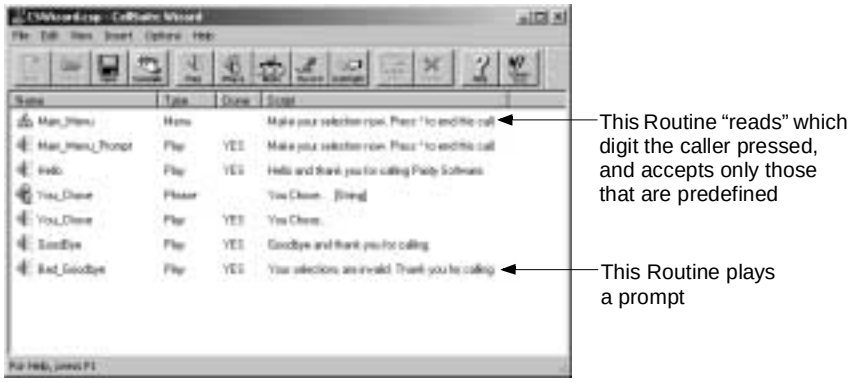


Figure 4-4 CallSuite Wizard—main window

Each item in the main window represents a Routine included in your VBSampleOutbound application. Let's view the properties of the Main Menu Routine.

2. From the CallSuite Wizard main window, double-click on Main_Menu to display the Menu Routine Editor.

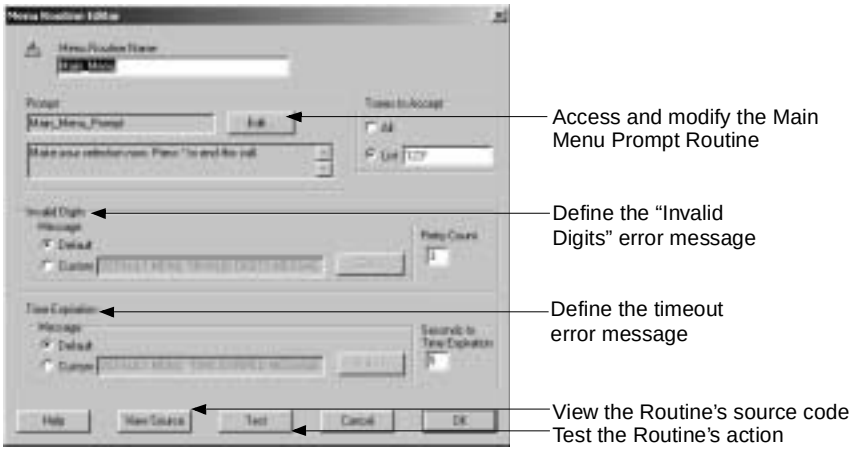


Figure 4-5 Menu Routine Editor dialog

From this dialog we can access the Main Menu Prompt Routine and modify it.

3. Click Edit.

The Play Routine Editor dialog displays. You can select another recording or create one of your own. We're going to create a new recording.

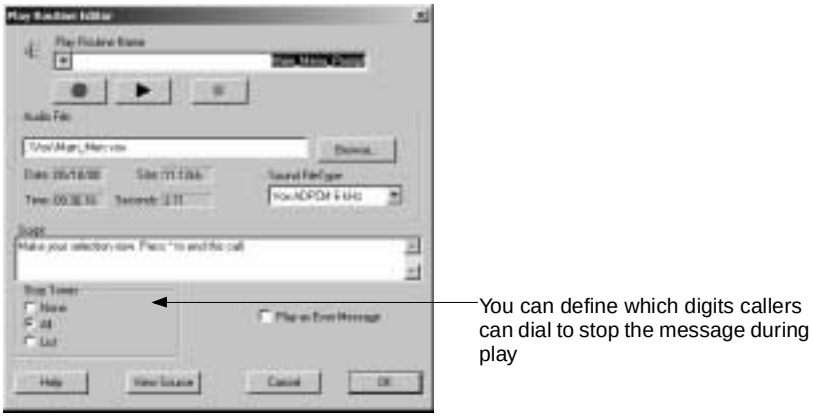


Figure 4-6 Play Routine Editor dialog

4. In the Script field, type **Press 1 to hear our stock price or press 2 to reach Technical Support. Press star to end this call.**

   SimPhone lets you record by using a microphone attached to your sound card.

5. Click Record (red dot) and record the message you typed. Then click Stop (black rectangle) when you are finished.
6. Review the message by clicking Play (the black triangle), and make new recordings until you are satisfied. When you are finished, click OK to close this dialog.
7. From the CallSuite Wizard main window, choose File | Save.

Step 2: Modifying the Hello Routine

Next, we want to change the Hello Routine so that it plays a different company name:

1. Double-click the Hello Routine to display the Play Routine Editor dialog.
2. In the Script field, type **Hello, and thank you for calling Precision Software.**
3. Click Record, say this message, and click Stop when you're finished. Review the message by clicking Play. If you're not happy with your message, make new recordings until you are satisfied.
4. Click View Source to see your changes reflected in the Routine's source code.

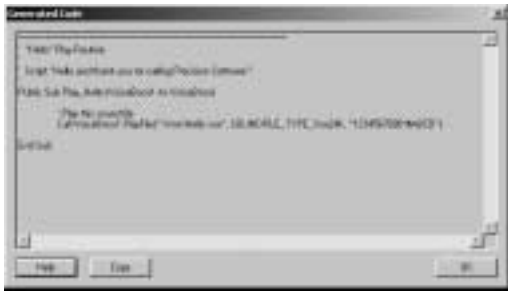


Figure 4-7 Generated Code dialog

5. Click OK to close the Generated Code dialog, and then click OK again to close the Play Routine Editor dialog.
6. From the CallSuite Wizard main window, choose File | Save.

Step 3: Editing and Creating Routines for the Menu choices

In Tutorial 1 we selected two menu choices for callers to choose from. Now we need to record messages for those choices. First, let's edit the Play Routine for the Technical Support menu option.

1. From the CallSuite Wizard main window, click the Play Routine to display the Play Routine Editor dialog.
   a. In the Play Routine Name field, type **Tech_Support**.
   b. In the Script field, type **You chose Technical Support.**
   c. Record this message, then click OK to save your changes and close the dialog.

Now let's create a Phrase Routine for the Stock Price. In order to play a message that contains variable data, like dates or dollar amounts, you need to build the elements of the phrase. When the caller chooses to listen to the stock price, a message like this plays:

"Our company's stock price was $256.84 May 16, 1998."

The elements in this phrase consist of: two Play Routine— "Our company's stock price is" and "on" , and two variables—a Date variable and a Money variable.

2. From the CallSuite Wizard main menu, double-click the "You Chose" Phrase Routine.

   The Phrase Routine Editor displays.

3. In the Phrase Routine Name field, type **Stock_Quote**.

   Note, if you type a space in this field, CallSuite Wizard automatically inserts an underscore in its place.

4. In the Phrase Elements window, select the String element ABCD and click Remove Element to delete it.
5. In that same window, double-click the Play Routine called “You Chose.”

   The Play Element Editor dialog displays.

   a. In the Play Routine Name field, type **On**.

   b. In the Script field, type **On**.

   c. Record this message, and then click OK to close the Play Element Editor.
6. From the Phrase Routine Editor, click Date, then double-click on the new Date element.

   The Insert Date Element dialog appears, showing this format: 20011010.

   a. Click Play to hear how the date is said during play time, then click OK to close this dialog.
7. Click Play Routine.

   The Play Element Editor dialog displays.

   a. In the Play Routine field, type **Our_Stock_Price**.

   b. In the Script field, type **our company’s stock price is**.

   c. Record this message, then click OK to save your changes and return to the Phrase Editor.
8. Finally, click Money to insert a Money element.

   In the Script field at the bottom of the Phrase Routine Editor dialog, you can see the order in which the elements play during play.

Figure 4-8 Phrase Routine Editor dialog

9. Click Play from the Play Routine Editor to hear the Phrase with a sample date and price.
10. Click OK to return to the main window in the CallSuite Wizard, and then choose File | Save.

Step 4: Editing the Goodbye Routine

Now we want to change the Goodbye message so that our company name is mentioned again.

1. From the CallSuite Wizard main window, double-click on the Goodbye Routine.

   The Play Routine Editor displays.

2. In the Script field, type **Goodbye, and thank you for calling Precision Software.**
3. Record the message you typed. When you are finished, close the Play Routine Editor.

Step 5: Changing the Bad Goodbye Routine

If the caller repeatedly presses the wrong digit or no digit at all, the Bad Goodbye message plays and then the application hangs up. We want to change the Bad Goodbye message so that it matches your voice like the other prompts.

1. Double-click on the Bad Goodbye Routine.

   The Play Routine Editor displays.

2. In the Script field, type **Your selections are invalid. Thank you for calling Precision Software.**
3. Record the message you typed. When you're finished, close the dialog.
4. From the Visual Basic main window, choose File | Save Project.

Step 6: Changing the Default Error Prompts

Callers will hear error messages if they press digits that are not predefined to perform a Routine or if they press no digits at all. If an invalid digit is pressed, this Menu Invalid Digit message plays:

"That is not a valid response."

If no digits are pressed at all, a Menu Time Expired message plays:

"No response received."

Both of these are "default error messages" that were automatically included in the generation of your application. We want to rerecord these messages so that they play in your voice.

We are going to rerecord the Menu Invalid Digit message first.

1. From the Visual Basic main window, choose Tools | CallSuite Wizard v8 to display the CallSuite Wizard.
2. From the CallSuite Wizard main window, choose Options | Default Messages | Menu Invalid Digit Message.

   The Routine Editor dialog for this Routine displays. We still want the script to say:

   "No response received."

   —so we do not change the script text, but we do want the message to play in a different voice.
3. Record the message in the script field, then click OK to save your changes and close the dialog.

Now we are going to rerecord the "No response received." message.

4. From the CallSuite Wizard main window, choose Options | Default Messages | Menu Time Expired Message.
5. The Routine Editor dialog for this Routine displays.
6. Rerecord the message in the Script field, then click OK to save your changes and close the dialog.
7. Choose File | Save to save your changes and then close the CallSuite Wizard main window.

Testing Your Application

Now it's time to test your application.

1. From the Visual Basic main window, choose Run | Start.

   Visual Basic compiles the application, and then runs it. The SimPhone main window and the Outbound Telephony Application dialogs display.
2. Click Dial from the Outbound Telephony Application dialog.
3. On the Call Progress Analysis Results dialog, click Connect.

   You should hear your newly recorded welcome message and prompt asking you to make a selection.
4. Dial 1 from the SimPhone main window to hear the response in your voice.
5. This time, when you are prompted, dial 9 to hear the invalid response message in your voice. Then press star to end the call.
6. When you are finished, choose Run | End from the Visual Basic main window and close Visual Basic.

Using Graphical VOS 5

This chapter provides an overview of Graphical VOS FlowCharter and two tutorials in which you create applications using FlowCharter. It contains the following sections:

- FlowCharter Overview
- Tutorial 1: Creating a Basic Telephony Application
- Tutorial 2: Creating an IVR Application

FlowCharter Overview

The Graphical VOS FlowCharter is a graphical tool that lets you build CT applications using a flow chart layout. This kind of layout lets you easily see the organization and flow of your application.

To see the Graphical VOS IDE main window, display the Inbound sample application from the Start menu by choosing Programs | Parity Software | Graphical VOS | Samples | FlowCharter Samples | Inbound. Then double-click on the Main.flw file in the Project window.

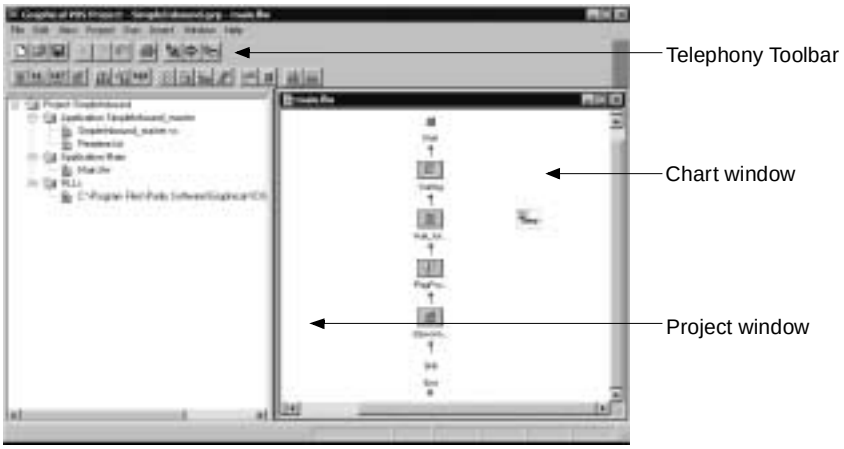


Figure 5-1 Graphical VOS IDE—main window

On the left side of the main window, the Project window displays a tree of files that make up the Inbound sample application. The Chart window on the right displays a flow chart view of the Inbound application.

When you develop a new application, you must create a new project first. A new project tree displays on the left side of the Graphical VOS main window. Within this project you can create telephony applications using FlowCharter to create flow charts or VOS source code. Applications display in the project tree on the left side of the main window and as flow charts or source code on the right side.

Flow charts consist of linked icons called cells. Cells are modules of code that perform specific telephony functions. Your application executes cells in the order they are linked in the flow chart. You can make changes directly to the chart by moving the cells, or by changing the flow of the links connecting them.

Each cell has properties that include its name, what it does, and any parameters you need to define. Although every cell has default properties that you can use, you can also modify them to suit your needs. You add cells to your chart by dragging them from one of the telephony function toolbars.

Using Toolbars

There are seven toolbars that you can use to build, edit, and debug your applications.

Standard—Use the three rightmost cells on this toolbar to compile, run, and debug your project. This toolbar displays by default when you open Graphical VOS.

Chart—Use the Chart cells to manage the cells on your flow chart.

Telephony—Use the Telephony cells to add functions such as waiting for a call, dialing digits, receive digits dialed

Mailbox Functions—Use the Mailbox Functions cells to access and manage voicemail.

Fax Functions—Use the Fax Functions cells to send and receive faxes.

Conferencing Functions—Use the Conferencing Functions cells to set up and administer conference calls.

MSI Station Functions—Use the MSI Station Functions cells to control MSI stations.

NetHub Plus—Use the NetHub Plus cells to add and edit messaging functions.

View the Telephony toolbar by choosing View | Toolbars | Telephony from the Graphical VOS main window.

Figure 5-2 Telephony Toolbar in Graphical VOS

Flow Charts

When you're developing your application, your flow chart has a New cell that acts like a cursor, showing where the next cell will be inserted by default. Each time you insert a cell, the New cell moves to the next place that you're most likely to insert a cell. This saves you from having to drag individual cells to your chart and significantly reduces the number of mouse clicks and keystrokes required to create an application. If you want, you can also drag the New cell anywhere on the screen.

When you run your application, execution begins at the Start cell and then continues to subsequent cells in the order they are arranged on the chart.

Connecting Cells

The cells on your chart are connected by links containing arrows that show the flow of your application. A stub is a link that is connected to only one cell instead of two and appears as a red X below a cell.

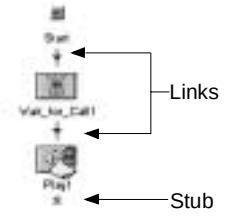


Figure 5-3 Flowcharter cells showing links and stubs

When you insert a new cell directly below a stub, FlowCharter automatically links the two cells. You can check a link's properties by right-clicking the link and selecting Properties, or by double-clicking the link. From the Properties dialog, you can hide a link (it appears as a green arrow, but won't change the flow of your application) or connect it to any other cell on the chart. You can also drag links to other cells on the chart.

Links enter a cell at any one of three sockets on the top or sides and leave from its outlet at the bottom of the cell.

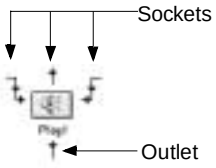


Figure 5-4 Flowcharter cell showing sockets and an outlet

Cell Properties

Open the properties for a cell by double-clicking the cell or clicking it once and pressing the Enter key. You can also view the properties by right-clicking the cell and selecting Properties or by pressing Ctrl+Enter.

The Cell Properties dialog appears with tabs, and each tab defines a property for the cell. For example, the Cell Name tab displays the name of the cell. The What Am I? tab displays a detailed description of the code's function. (This tab is read only, but you can modify the information on most tabs.) Just as cells have functions that differ, they may also have more or fewer tabs on which you can specify properties that apply to that particular cell.

If a cell has properties that still need to be set, it appears with a little yellow note tacked to it. This is called a *To Do note*. In this case, the Cell Properties dialog has a Things To Do tab that outlines which properties you need to set.

Figure 5-5 Start Chart Cell Properties dialog

In the next two sections we've provided tutorials to give you a chance to create applications using FlowCharter.

Tutorial 1: Creating a Basic Telephony Application

In this tutorial you'll create a very simple application that answers an incoming call, plays a message to the caller, and then hangs up.

Before you Begin

Make sure Graphical VOS has been installed correctly. If you have questions or want more information on any of the FlowCharter cells or VOS code you see during this tutorial, see the Graphical VOS online help.

Step 1: Creating a Graphical VOS Project

1. Start SimPhone and Graphical VOS from the Windows Start menu.

2. In the Welcome to Graphical VOS dialog, select Create a new project and click OK.

Figure 5-6 Welcome to Graphical VOS dialog

3. In the New dialog:
   a. Make sure that Project is the selected File Type.
   b. In the File name field, type **Tutorial**.
   c. In the Location field, type this path for the project:
      ```
      C:\Program Files\Parity Software\Graphical VOS\Tutorial\FlowCharter
      ```
   d. Click OK.

Figure 5-7 New dialog—displays when you choose Create a new project button

Graphical VOS creates your project. In the Project window, you see a tree on the left side that shows the project's files: a master application and an RLL. To find out more about RLLs (Runtime Link Libraries) see the Graphical VOS online help.

Step 2: Adding a Flow Chart to the Project

1. From the main menu in Graphical VOS, click File | New.
2. In the New dialog:
   a. Make sure that Chart is the selected File Type.
   b. In the File name field, name the chart **MyFirst**.

c. Select the Add to Project checkbox, and then click OK to create the new chart.

The chart you created contains a Start cell, a New cell, and an End Chart cell.

Step 3: Making the Telephony Toolbar Visible

1. From the main menu in Graphical VOS, choose View | Toolbars | Telephony.

   The Telephony Toolbar displays.

2. Click the Wait for Call cell icon on the Telephony Toolbar.

   The Wait for Call cell appears on your chart and automatically links itself to the Start cell. When you run your application, the Wait for Call cell waits for an incoming call. When a call arrives, the cell answers the call and passes control to the next cell.

   Notice that the New cell has moved below the Wait for Call cell.

3. Click the Play cell icon on the Telephony Toolbar.

   The Play cell appears on your chart and automatically links itself to the Wait for Call cell. When you run your application, the Play cell plays a prompt over the phone line and passes control to the next cell.

   You can also insert cells by dragging them from the toolbar.

4. Click the Disconnect cell on the Telephony Toolbar.

   The Disconnect cell appears on your chart and automatically links itself to the Play cell. At the end of a phone call, the Disconnect cell hangs up the phone line.

5. To complete your application, drag the existing End Chart cell up one space and drop it where the New cell sits.

   The New cell is bumped down one spot on the chart and the End Chart cell automatically links itself to the Disconnect Call cell. When you run your application, the End Chart cell restarts the application and execution resumes with the Start cell.

   Now you're ready to run your application!

Step 4: Running Your First Application

1. From the Run menu in Graphical VOS, choose Run MyFirst.flw.

   Graphical VOS compiles your application and starts VOS. If you have trouble, click View | Runtime Log and consult the online help for troubleshooting information.

2. In SimPhone, click Ring to simulate a call to the voice card. If you didn't start SimPhone at the beginning of this tutorial, launch it from the Windows Start menu now.

   Your application answers the call and plays a prompt before hanging up. You can make another call if you want, and your application will answer and play the message again.

3. From the Graphical VOS Run menu, choose Stop Running to stop the application.

Step 5: Setting Play Cell Properties

There is a little yellow To Do note tacked to the Play cell so you still need to set properties for the Play cell. Because you didn't set them before running the application, the cell used its default behavior.

The Properties dialog shows all the properties associated with the cell. On the Things To Do tab, you see that *Select a prompt* is a To Do item. This means that the Play cell is defined, but the prompt to play has not been specified. As you saw, the Play cell still runs, but because no prompt has been defined it plays the default prompt.

To specify a new prompt:

1. Right-click the Play cell and select Properties from the drop-down menu.
2. On the Play Cell Properties dialog, click the Prompt tab.

   If prompts had already been defined, you could select one from the Prompt Name list.

3. No prompts have been defined in this project so click Prompts to display the Prompts dialog.
4. Click New to display the New Prompt - Properties dialog, then complete these steps:
   a. In the Prompt Name field, type **Hello**.
   b. Click Browse and locate the following audio file that has been recorded for this tutorial:
      `C:\Program Files\Parity Software\Graphical VOS\Tutorial\Prompts`
   c. Select Hello.vox, and then click Open.

d. Select 24 kbps 4-bit 6 kHz ADPCM VOX for the File Type.

e. For the script, type: **Hello and thank you for calling Parity Software**.

f. Click OK to close the Prompt Properties dialog, then click OK again to close the Prompts dialog.

Now you can see the Play cell properties. Note that the prompt you just defined has been selected for this cell.

5. Click OK to close the Play Cell Properties dialog.

   Notice that the note is gone from the cell.

6. Run your application again to hear the new prompt.
7. When you're done, choose Stop Running from the Graphical VOS Run menu to stop the application.

Tutorial 2: Creating an IVR Telephony Application

In this tutorial you are going to create an IVR (Interactive Voice Response) telephony application that:

- Answers an incoming call
- Plays a menu that gives the caller three options:
  - — Leave a message
  - — Enter a phone number
  - — Hear the company's stock price
- Performs the action the caller requested, or plays a goodbye message if the caller didn't make a valid choice.
- Hangs up the phone line

Before You Begin

This tutorial extends Tutorial 1: Creating Your First Telephony Application. If you haven't completed all of the steps in Tutorial 1, go back and finish them before beginning Tutorial 2.

Step 1: Adding a New Flow Chart to Your Project

1. If you haven't already done so, start Graphical VOS and open the Tutorial project you created in Your First Telephony Application.

2. From the main menu, click File | New.
3. In the New dialog:
   a. Make sure that Chart is the selected File Type.
   b. In the File name field, name the chart: **AutoAttendant**.
   c. Make sure the Add to Project checkbox is checked.
   d. Click OK to create the new chart.

FlowCharter adds the new chart to your project. In the Project window, you can see three applications listed:

Tutorial_master—The master application controls which files are launched when you run your project. Graphical VOS creates and maintains the master application. You shouldn't delete this file, and in most cases you won't need to modify it.

MyFirst—This is the application you created in Getting Started: Your First Telephony Application.

AutoAttendant—This is your new application.

Step 2: Configuring the Trunk 0

To specify which application to run:

1. From the main menu, choose Project | Configure Trunks.
2. In the Trunk Configuration dialog, click Add to open the Configure Trunks dialog.
3. Set both Trunk Range fields to 0.
4. Set the Program Type to Fixed, and then select AutoAttendant from the Program drop list.

Figure 5-8 Configure Trunks dialog

5. Click OK to close the Configure Trunks dialog.

In the VOS Trunk Configuration dialog, you can see that Trunk 0 has been set.

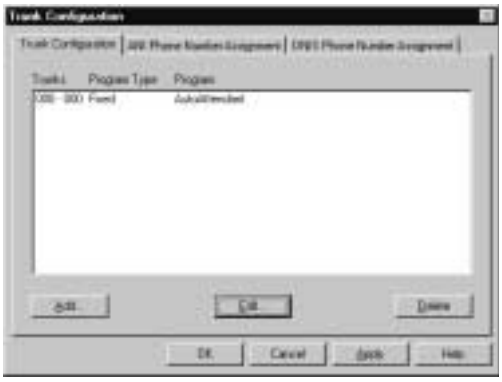


Figure 5-9 Trunk Configuration dialog

Each time a call comes in on Trunk 0, that call is sent to the AutoAttendant program.

6. Click OK to close the Configure Trunks dialog.

Step 3: Waiting for an Incoming Call

When the application starts, we want it to wait for and answer an incoming call.

Click the Wait for Call cell icon on the Telephony toolbar to insert a Wait for Call cell.

Now the application waits for an incoming call, and takes the phone off hook when a call comes in. The New cell moves below the Wait for Call cell, where we'll insert the next cell.

Step 4: Adding a Menu

Next, let's add a menu to our application. Most call processing applications begin with a main menu similar to this one:

"Hello and thank you for calling our company. For this option, press 1, for that option, press 2..."

1. Click the Menu cell icon on the Telephony toolbar to insert a menu.

   A dialog box displays, and now you can select the tones to use in the menu.

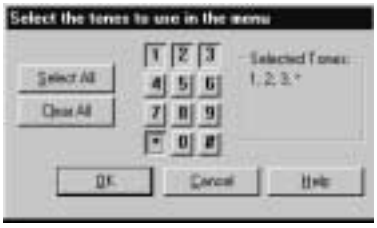


Figure 5-10 Select the tones to use in the menu dialog

We want our application to let the caller leave a message, enter a phone number, or hear the current stock price, so we need to create a menu that has three options. We also want the caller to be able to press the star key to exit the call.

2. Select the 1, 2, 3, and * buttons—the tones you select display in the Selected Tones list—then click OK.

The note attached to the Menu1 cell means that there are properties you need to set for the cell.

3. Double-click the Menu1 cell to see its properties.
4. Click the Things To Do tab, and you'll see that you need to select a prompt for the menu that lets callers know how to access the different menu options.
5. Click the Prompt tab.

   In the Prompt Name list box, you'll see that the flow chart doesn't have any prompts available so you need to set up a prompt for the menu.

6. Click Prompts, and then click New to open the New Prompt - Properties dialog:

   a. In the Prompt Name field, type **MainMenu**.

      Next, you need to specify which sound file to play—main_men.vox has been provided for this tutorial:

   b. Click the Browse button to locate and select the Main_men.vox sound file.

      By default, you'll find it in the following directory:

      ```
      C:\Program Files\Parity Software\Graphical VOS\Tutorial\Prompts
      ```

   c. Click Open to return to the New Prompt - Properties dialog and then from the File Type list, select 24 kbps 4-bit 6 kHz ADPCM VOX.

   d. In the Script field, type:

      To leave a message, press 1. To enter your phone number for a return call, press 2. To hear today's stock price, press 3. To end this call, press star.

Figure 5-11 New Prompt - Properties dialog

e. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Menu Cell Properties dialog.

   The MainMenu prompt is now listed in the Prompt Name field.

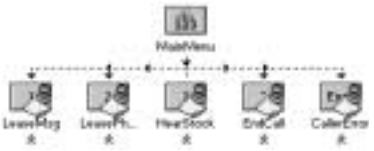
7. Click the Cell Name tab. In the Cell Name field, name the cell **MainMenu**.
8. Click OK to return to your flow chart.

   The MainMenu cell no longer has a note attached.

   Depending on what the caller dials after hearing the menu prompt, the application flows to one of the Case cells below the Menu cell. Each Case cell shows the number that the caller must press to choose that option. Next we'll give these Case cells more descriptive names.

9. Double-click the first cell's name, Choice1_1. The cell name is highlighted, and you can type another name in its place. Name this cell **LeaveMsg**.

10. When you are done typing, click anywhere in your chart and the cell is renamed. Repeat this process and rename the other Case cells **LeavePhoneNum**, **HearStock**, **EndCall**, and **CallerError**.



Next, you need to add cells to let the caller leave a message, enter his or her phone number, hear the stock price, or end the call.

Step 5: Adding a Record Cell

If the caller presses 1 from the main menu, the application records a message. After the caller finishes recording a message, we want the application to hang up and end the call.

1. Drag the Record cell icon from the Telephony toolbar to below the LeaveMsg Case cell.
2. Drag the Disconnect cell icon from the Telephony toolbar to below the Record1 cell, and then drag the link at the bottom of the Disconnect1 cell to the End Chart cell.
3. Double-click the Record1 cell to see the cell's properties.

   The Things To Do tab tells you that you need to enter a file name and a file type and select a prompt for the cell.
4. Click the File to Record tab and type **Message.vox** in the File Name field.
5. From the File Type drop-down list, select 24 kbps 4-bit 6 kHz ADPCM VOX.

   To set up a prompt for the Record cell, follow the same procedure that you did for the Menu.
6. Click the Prompt tab, then click the Prompts button to open the Prompts dialog.
   a. Click New, and in the Prompt Name field, type: **LeaveAMessage**.
   b. Click Browse and select Leave_a_.vox in the Select a Sound File dialog.
   c. Click Open to return to the New Prompt - Properties dialog. From the File Type list, select 24 kbps 4-bit 6 kHz ADPCM VOX.
   d. In the Script field, type:

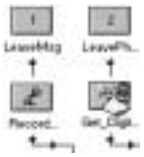
      Please leave a message at the tone. When you are finished, press pound.
   e. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Record Cell Properties dialog.
7. Click the Cell Name tab. In the Cell Name field, type **RecordMessage**.
8. Click OK a final time to return to your flow chart.

Step 6: Adding a Get Digits Cell

If the caller presses 2 from the main menu, we want to let him or her enter a phone number. After the caller has entered a phone number, we want to end the call.

1. Drag the Get Digits button on the Telephony toolbar to below the LeavePhoneNum Case cell.

2. Drag the link at the bottom of the Get_Digits1 cell to the Disconnect cell.



3. Double-click the Get_Digits1 cell to see its properties.

   The Things To Do tab says that you need to select a prompt for the cell.

4. Click the Prompt tab, then click the Prompts button to open the Prompts dialog.

   a. Click New and then in the Prompt Name field, type: **EnterYourPhoneNumber**.

   b. Click Browse and select Enter_yo.vox in the Select a Sound File dialog. Click Open to return to the New Prompt - Properties dialog.

   c. From the File Type list, select 24 kbps 4-bit 6 kHz ADPCM VOX.

   d. In the Script field, type:

      Enter your phone number. Press pound when you are finished.

   e. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Get Digits Cell Properties dialog.

   We also need to set up the number of digits to get from the caller, which you specify on the Digit Limits tab. To have a valid phone number, we need between seven and ten digits from the caller. We'll give the caller 30 seconds to enter all of the digits.

5. Click on the Digit Limits tab:

   a. Type **7** in the Get at least __ digit[s] field.

   b. Type **10** in the Get at most __ digit[s] field.

   c. Type **30** in the Allow up to __ seconds to get all digit[s] field.

Figure 5-12 Get Digits Cell Properties dialog

6. Click the Cell Name tab. In the Cell Name field, type **EnterPhoneNumber**.
7. Click OK to return to your flow chart.

Step 7: Adding a Phrase

If the caller presses 3 from the main menu, we want to play a message that tells the current stock price. In order to play a message that contains variable data, like dates or dollar amounts, you need to create a phrase. If the caller chooses the stock price option, he or she hears a sentence like this:

"Our company's stock price is $216.84."

This phrase is made up of one Play cell that says *Our company's stock price is*, plus one Phrase cell that is a Money variable. After the caller has heard the stock price, we want to end the call.

1. Drag the Play cell icon from the Telephony toolbar to below the HearStock Case cell.

   Now you need a Phrase:

2. Next, drag the Phrase cell from the Telephony toolbar to just below the Play1 cell.

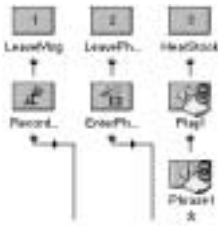


Figure 5-13 Flowcharter—adding a Phrase cell

3. Drag the link at the bottom of the Phrase1 cell to the Disconnect cell.
4. Double-click the Play1 cell to see the cell's properties.

   The Things To Do tab tells you that you need to select a prompt for the cell.

5. Click the Prompt tab, then click the Prompts button to open the Prompts dialog:
   a. Click New and in the Prompt Name field, type **StockPrice**.
   b. Click Browse and select Stock_pr.vox in the Select a Sound File dialog, then click Open to return to the New Prompt - Properties dialog.
   c. From the File Type list, select 24 kbps 4-bit 6 kHz ADPCM VOX.
   d. In the Script field, type **Our company's stock price is**.
   e. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Play Cell Properties dialog.

6. Click the Cell Name tab and in the Cell Name field, type **CompanyStockIs**. Click OK to return to your flow chart.
7. Double-click the Phrase1 cell to see that cell's properties.

   The Things To Do tab tells you that you need to enter a value to speak and specify how that value should be spoken.
8. Click the Type tab and select Money from the Speak as field.
9. Next, click the Value to Speak tab:

   In most applications, you would read this value from a variable or database, but to keep this tutorial simple, we'll just assign a dollar amount to read.

   a. In the Type field, select Expression.

   b. Type **"216.84"** in the Expression field.

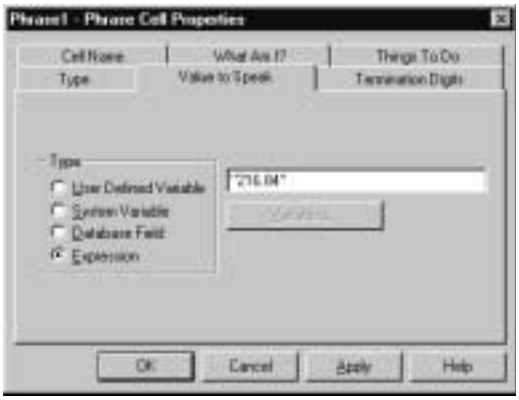


Figure 5-14 Phrase Cell Properties dialog

10. Click the Cell Name tab and in the Cell Name field, type **Price**. Click OK to return to your flow chart.

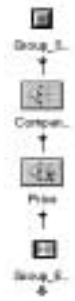
Step 8: Placing the Phrase in a Group

A Group helps you organize your flow charts by representing a set of related cells using a single cell. This shrinks the overall size of the flow chart, making it easier to see it as a whole. You can expand the set of cells in a Group cell by clicking on it.

1. Select the CompanyStockIs and Price cells by clicking one and then holding down the Ctrl key while you click the other.
2. From the main menu, choose Edit | Create Group.

   The Group1 cell appears in place of the cells you selected.

3. To rename the Group1 cell, right-click it and choose Properties from the right-click menu, then click the Cell Name tab and type **StockPrice** in the Cell Name field. Click OK.
4. Double-click the StockPrice cell, and you'll see the CompanyStockIs and Price cells between a Group Start cell and a Group End cell.

   Similar to the Start cell and End Chart cells, the Group Start and Group End cells mark the beginning and end of the Group.
5. Click the Close button in the upper right corner of the AutoAttendant1:StockPrice window to close the Group and return to the flow chart.

Step 9: Adding a Thank-You Message

If the caller presses * from the main menu, we want to thank him or her for calling, and hang up.

1. Drag the Play cell icon from the Telephony toolbar to below the EndCall Case cell.

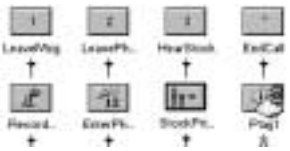


Figure 5-15 FlowCharter—adding a Play cell

2. Drag the link at the bottom of the Play1 cell to the Disconnect cell, then double-click the Play1 cell to see the cell's properties.

   The Things To Do tab says that you need to select a prompt for the cell.
3. Set up a prompt for this Play cell:
   a. Name the prompt **ThankYou**.
   b. Select Goodbye.vox as the sound file.
   c. Select 24 kbps 4-bit 6 kHz ADPCM VOX for the File Type.
   d. In the Script field type **Thank you for calling. Goodbye.**
4. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Play Cell Properties dialog.
5. Click the Cell Name tab. In the Cell Name field, type **ThankYou**. Click OK to return to your flow chart.

Step 10: Adding an Error Message

In the main menu, if the caller fails to enter a valid choice after three tries, the application flows to the CallerError Case cell. We want to play a special message in this case and then end the call.

1. Move the New cell below the CallerError Case cell, and click the Play cell icon on the Telephony toolbar.
2. Drag the link at the bottom of the Play cell to the Disconnect cell.
3. Double-click the Play cell to see the cell's properties.

   The Things To Do tab tells you that you need to select a prompt for the cell.
4. Set up a prompt for this Play cell:
   a. Name the prompt **Invalid**.
   b. Select Bad_good.vox as the sound file.
   c. Select 24 kbps 4-bit 6 kHz ADPCM VOX for the File Type.
   d. In the Script field type **Your choices were invalid. Goodbye.**
5. Click OK to create your new prompt and return to the Prompts dialog. Click OK again to return to the Play Cell Properties dialog.
6. Click the Cell Name tab. In the Cell Name field, type **Sorry**. Click OK to return to your flow chart.

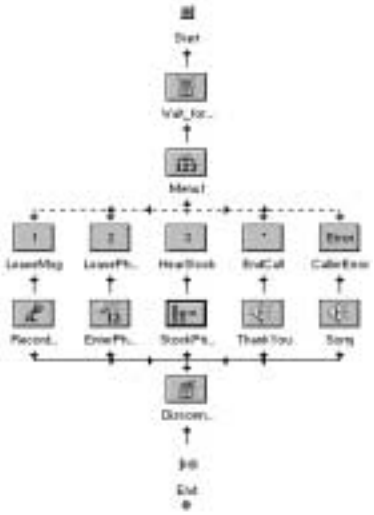


Figure 5-16 FlowCharter—your finished flowchart

7. From the Graphical VOS menu, choose File | Save.

Step 11: Running the Application

1. To run your application, select Run | Run autoattendant.flw from the main menu.

   Graphical VOS saves and compiles your application, then starts VOS.

2. In SimPhone, click Ring to simulate a call to the voice card.

   Your application answers and plays this prompt:

   "To leave a message, press 1. To enter your phone number for a return call, press 2. To hear today's stock price, press 3. To end this call, press star."

3. Enter a number between 1 and 3.

   — If you press 1, you will be prompted to leave a message.

   — If you press 2, you will be asked to enter a seven-digit number.

   — If you press 3, you will hear a phrase that says:
   "Our company's stock price is two hundred sixteen dollars and eighty-four cents."

   After the application completes the task that accompanies the option you chose, it hangs up. If you press the star key (*), the application plays a closing message and hangs up. You can repeat this as many times as you wish by making another call to the voice card.

4. When you are finished, exit the application by choosing Run | End from the Graphical VOS main menu.

VOS Language Concepts 6

This chapter gives you an overview of the most important concepts of the VOS programming language and contains the following sections:

- Overview
- Writing Simple Expressions
- Forming Complex Expressions
- Using Statements
- Constructing Loops
- Utilizing Functions

Overview

The VOS language is easy to learn, and you can create robust applications quickly. Specially designed for call processing, VOS uses very little dynamic memory allocation so live systems are very unlikely to run out of memory.

More simple to use than Visual Basic and C/++, VOS borrows some of the best ideas from the C language, but avoids many of the problems that make using C difficult for beginners (such as syntax) and experts (such as data types and dynamic memory allocation).

If you are a beginner, you'll find the VOS language easy to learn; and if you're an expert who knows C, you'll quickly adapt to the changes relative to C (which are designed to help beginners avoid common mistakes).

Source Code Basics

As you write VOS programs, keep the following rules in mind:

Differentiate Between Upper- and Lower-case Letters

VOS is a case-sensitive language. All parts of the language recognize the difference between upper-case letters (ABC...) and lower-case letters (abc...).

Begin Comments with a Pound Sign (#)

You can add comments to your code by typing a pound sign (#) at the beginning of a line. Comments continue up to the end of that same line.

The pound sign may appear as £ or another special character on non-US PCs; it is the ASCII character with code 35 decimal, 23 hex.

Use End-of-line Marks (;)

The end-of-line marks have no syntactical significance except that they terminate a line of code.

Multiple statements may be included in a single line, although this is discouraged because it generally makes the source code harder to read.

You can find more detailed information about the VOS programming language in the Graphical VOS online help.

The basic structure of VOS flows like this:

Example 6-1 Source code example

```
# This program is called inbound.vs.
  program
   TrunkWaitCall();
   TrunkAnswerCall();
   if (TrunkGetState() strneq "Connected")
      voslog("Unexpected trunk state ", TrunkGetState());
      stop;
   endif
   InboundCall();
   TrunkDisconnect();
   restart;
endprogram
```

Writing Simple Expressions

Simple expressions consist of a single value. There are three kinds of simple expressions: values, variables, and constants.

Values

All values in VOS are stored as character strings. Character strings are written as a sequence of characters inside double quotes:

```
"This is a string"
```

Numerical Values

Numbers are represented by strings of decimal digits. For example, one hundred and twenty-three is represented as the string of three characters “123.” You can omit the double quotes when writing a number, so both of these examples are acceptable ways of writing numbers:

```
"123"

123
```

Logical Values

True and False values are also represented as strings. False is represented as an empty string containing no characters, written like this:

```
""
```

True is represented as a string containing the decimal digit 1:

```
"1"
```

Variables

A variable has a name and contains a character string. All variables are set to empty strings when VOS starts and when a restart statement is executed. Variables must be declared, meaning that you must give them a name and maximum length before VOS can use them. The maximum length cannot exceed 127 characters or VOS truncates it to equal 127 characters. You can declare variables in two places: before the start of the main program, or inside of a function.

If you are running VOS in Debug mode, a warning message is issued if a string is truncated by assigning it to a variable. To avoid this warning, assign the string to a variable using a substr function, which is a VOS function that returns a substring of a given string.

Why a maximum length for each variable? Why doesn’t VOS allocate memory dynamically? This is because call processing systems must often run unattended for days, weeks, and months at a time. VOS is designed to avoid dynamic memory allocation in all areas to avoid problems caused by memory fragmentation and running out of memory in a live system. This is one of many reasons that we chose to design a new language for VOS rather than use an existing language.

 *Caution:*

One of the most common programming mistakes that VOS beginners make is forgetting that variables have a fixed length and that values longer than the fixed length are truncated. This is another good reason to make a habit of using VOS

in Debug mode and to look at the VOS1.LOG file on a regular basis to make sure that there are no error messages or warning messages.

Example 6-2 Program with a variable declaration block

In this example, two variables are declared: one named x, with a maximum length of two characters, and one named y, with a maximum length of three characters.

```
dec

   var x : 2;

   var y : 3;

enddec


program

   x = 12;

   y = 123;

   y = "Too long"; # y becomes "Too"

endprogram
```

A variable name must start with a letter and may continue with any number of letters, digits (0 through 9) and underscore characters (_). Because the VOS language is case sensitive, the variable names X and x refer to two different variables.

Constants

A constant is a named value. That is, a name assigned to a fixed string of characters. Constants are declared inside a dec...enddec block like this:

```
dec

   var x : 2;

   var y : 3;

   const LINE_NUMBER = 1;

   const MAX_TIME = 60;

enddec
```

The name of a constant may be used anywhere a value is expected in the language. For example, a constant may be used to specify the maximum length of a variable:

```
dec

   const MAX_DIGITS = 7;

   var PhoneNumber : MAX_DIGITS;

enddec
```

Forming Complex Expressions

You can create more complex expressions using operators. Operators take one, two, or three values and produce a single value as a result. There are two kinds of operators, arithmetic operators and logical operators.

Arithmetic Operators

A familiar example of an operator is the plus sign (+). This is called the addition operator and it combines its left-hand value and its right-hand value to give a single result. VOS includes a set of five arithmetic operators. Similar to algebra, operators have different strengths called *precedences*.

Operation	Expression
Add	`Left + Right`
Subtract	`Left - Right`
Multiply	`Left * Right`
Divide	`Left / Right`
Change sign	`-Right`

For example, multiplication is stronger than addition, so in the expression A+B*C, the multiplication B*C is performed first and the result added to A. Parentheses () may be used to create groups and force evaluation in the desired order, so in the expression (A+B)*C, the addition will be performed first.

 *Caution:*

Arithmetic is performed internally by VOS using 32-bit integer precision. Results involving 10 or more decimal digits may be incorrect. No warning or error is given by VOS when a value overflows 32-bit precision.

Because VOS uses 32-bit integers internally, values from -2^{31} to $2^{31}-1$ can be represented, which is the range:

```
-2147483648 to 2147483647
```

This means all 9-digit decimal integers can be represented, but most 10-digit values cannot. For example, that you can always add two 9-digit numbers, so this works correctly:

```
999999999 + 888888888
```

However, this doesn't work because the intermediate multiplication result overflows 32 bits:

```
(999999999*5)/4
```

Arithmetic Using Decimal Points

VOS has no built-in functions for dealing with fixed-point or floating-point decimal arithmetic. The Fixed Point Math RLL, which is included in the default installation for VOS, provides decimal arithmetic functions fp_add, fp_sub etc. However, VOS applications can still work easily with decimals such as dollars and cents without needing to load an RLL. The easiest way to do this is to multiply all values by 100 (this is for precision to two decimal places; if you need precision to three decimals, multiply by 1000, and so on). You can then add, subtract, and compare values using these new units (corresponding to cents, tenths of cents, etc.) and convert back to dollars and cents as necessary when you are done. In the case of money values, this corresponds to keeping all values in cents.

Example 6-3 Converting money values between cents only and dollars and cents

Let's suppose that TotalCents contains a money value as cents:

```
Dollars = TotalCents/100;

Cents = TotalCents - Dollars*100;
```

If you have two variables containing dollars and cents, they can easily be converted to a total cents value:

```
TotalCents = Dollars*100 + Cents;
```

If you have a string, say in a variable named Dec, containing a decimal point character, you can convert to dollars and cents like this:

```
PosOfPoint = strpos(Dec, ".");

if (PosOfPoint eq 0)

   Dollars = Dec;

else
```

```
        Dollars = substr(Dec, 1, PosOfPoint-1);
        Cents = substr(Dec, PosOfPoint+1);
    endif
```

Logical Operators

VOS includes three logical operators and eight comparison operators that are used to include, exclude, or compare values to give a logical result (True or False) result. True is represented as “1”, and False is represented as “”. Logical operators that combine two True/False values:

Operation	Expression
Logical AND	Left and Right
Logical OR	Left or Right
Logical NOT	not Right

Comparison Operators

Comparison operators that combine values and give a logical (True/False) result:

Operation	Expression
Greater than	`Left > Right`
Greater than or equal	`Left >= Right`
Less than	`Left < Right`
Less than or equal	`Left <= Right`
Equal numerically	`Left eq Right`
Not equal numerically	`Left <> Right`
Equal as string	`Left streq Right`
Not equal as string	`Left strneq Right`

 *Caution:*

There is a subtle difference between the equality of numbers and strings. Strings are converted to a number by taking all the characters up to the first non-digit. So this means that although “123”, “0123”, and “123ABC” are all equal numerically, they are different when compared as strings. This is important to remember when making menus using the pound or star keys because we are comparing zero with zero:

```
"*" eq "*" # This gives True
"#" eq "*" # This also gives True
```

Be sure to use streq, not eq, when comparing non-numeric values.

Logical conditions can include arithmetic operators and parentheses just like other expressions. In fact, VOS makes no distinction between logical conditions and other expressions except to convert the final result to a logical value rather than a numerical value. Logical conditions are used in if-statements and to control the following kinds of loops: for, while, and do...until.

For example, this is a logical condition used in an if-statement:

```
if (Year eq 1996 and Month > 6)
```

Assignments

The assignment operator = is a special kind of operator. The left side of the operator must be a variable or array element (see the Graphical VOS online help for more about arrays), and the right side is any expression.

Example 6-4 Assignment operator

The following are examples of valid assignments:

```
x = 1;

Month = 3;

MonthName = "March";

TotalSeconds = Hours*3600 + Mins*60 + Secs;
```

There are other assignment operators which can be useful: `+=`, `-=`, `*=` and `/=`. The one most often used is +=, which could be described as *add to*. For example, this adds 1 to n:

```
n += 1;
```

The expression on the right-hand side is evaluated, and the value is added to the variable on the left. In a similar way, `-=` is *subtract from*, and so on. Since adding and subtracting 1 is so common, further short-hands `++` and `--` are provided. They work like this:

x++	Add one to x, result is x before adding one.
++x	Add one to x, result is x after adding one.
x--	Subtract one from x, result is x before subtracting.
--x	Subtract one from x, result is x after subtracting.

You don't have to be concerned about the result if all you want to do is add or subtract one from a variable. For example, the two alternative statements have exactly the same effect:

```
x++;

++x;
```

The result matters if you use the result later in executing a statement:

```
x = 8;

y = x++;# y=8, x=9

x = 8;

y = ++x;# y=9, x=9
```

Notice that different values are assigned to y. If you find +=, ++, and similar expressions confusing, you can always achieve the same result by simply using =. Although if you're a C programmer and find them convenient, go ahead and use them.

String Concatenation

The & operator combines two strings by placing the characters on the right side after the characters of the left side. This is called string concatenation. For example, A & B gives AB. The following example results in FullName being assigned Joe Smith:

```
FirstName = "Joe";

LastName = "Smith";

FullName = FirstName & " " & LastName;
```

 *Caution:*

Note that string concatenation is one of the strongest operators. This can lead to subtle errors when used together with arithmetic operators. See the following example to find out how to avoid this kind of error.

Example 6-5 Using a string concatenation operator

This expression assigns 0 to x:

```
x = "Result is " & 2*2;
```

This is because & is stronger than *, so concatenation is done first, giving the result:

```
x = "Result is 2"*2
```

But *Result is 2* is zero when converted to a number (because the first character is not a digit), so the final expression becomes 0*2 = 0. To make sure that the multiplication is done first, you can use parentheses:

```
x = "Result is " & (2*2);
```

Using Statements

There are four basic kinds of statements: switch/case block statements, goto statements, jump statements, and include statements.

Switch/Case Statements

A *switch/case block* statement is a way to make a decision between several different options. If ... else ... endif is a one- or two-way decision, but switch/case allows for multiple decisions. A typical example where switch/case is useful is with touch-tone menus where the caller may press one of several different choices.

The basic idea behind a switch/case statement is that an expression is matched against several possible values. When a matching value is found, the following statements are executed.

Example 6-6 Switch/case statement

```
        Digit = sc_digits(line);

    switch (Digit)

    case 1:
```

```
        One();
    case 2:
        Two();
    case 3:
        Three();
    case "*":
        Star();
    default:
        Invalid();
    endswitch
```

One(), Two() ... Invalid() are functions which are defined elsewhere.

The expression in parentheses following switch is evaluated. The value is then compared with the expression following each case until a match is found. The first time a match is found, the statements following that case are executed until the next case, default, or endswitch is encountered. At that point, execution is transferred to the first statement following endswitch. If no matching cases are found, the statements following default are executed. If no matching cases are found and there is no default, then the entire switch...endswitch block is skipped.

Note that the comparisons are made as if streq had been used, not eq.

Full expressions, not just constants or variable names, are permitted following switch and following each case. As in other VOS language constructs, switch..endswitch statements may contain further switches, loops, etc. as required (although this is usually bad style—deep nesting usually results in hard-to-read code so we recommend using functions to break up such code into easy-to-read pieces).

Goto Statments

Goto statements are used to branch to a given location within your program. The target of a goto is a label, which is a name followed by a colon. A label may be applied to any statement.

Example 6-7 Goto statement

The following goto statement shows another way to compute a sum (Repeat and Done are labels):

```
        n = 1;
    Repeat:
```

```
        if (n > Max)
            goto Done;
        endif
        Sum = Sum + n;
        n = n + 1;
        goto Repeat;
    Done:
        vid_write("Sum = ", Sum);
```

The goto statement is written where *labelname* is defined somewhere in the same program section (main program or given function):

```
goto labelname ;
```

When a goto statement is executed, control is transferred to the statement with that label and execution continues from there. The label may be before or after a goto which references that label. You can't use a goto statement to branch from one function to another or from a function to the main program. You also can't apply a label without having a statement. For example, it is illegal to have a label immediately in front of an endfor keyword:

```
for (;;)
    goto EndOfLoop;
EndOfLoop:# Illegal!
endfor
```

Instead, you can achieve the desired effect by adding an empty statement that consists of just a semi-colon:

```
for (;;)
    goto EndOfLoop;
EndOfLoop:; # With semi-colon is OK
endfor
```

Jump Statements

Like Goto statements, *jump* statements are used to branch to a given location within your program. A jump statement transfers control to a special label in the main program. This may be done from anywhere in the program. If a jump is executed from within a function, the stack is cleared, which means that the memory of all function calls up to that point is erased. A jump label is defined using a label statement that may be applied to any statement in the main program:

```
label labelname :
```

The jump statement looks like this:

```
jump labelname ;
```

Jumps are useful for features like "*press star to return to the main menu,*" which can be very awkward to implement using other flow-of-control statements.

Include Statements

A source code file may include the complete contents of another file by using the *include* statement. This is typically used to include standard lists of constant declarations. For example, the keyword *include* is followed by the name of a file in double-quotes:

```
dec

   include "project.inc"

   var MyVar : 2;

enddec
```

An include keyword may occur anywhere in the source code, and is replaced by the contents of the referenced file. If the file plus.inc contains the single character +, then:

```
x = y include "plus.inc" z;
```

—is equivalent to:

```
x = y + z;
```

You can specify a search path for include files with the [IncludeFileDirs] section of the VOS Settings file.

Constructing Loops

You can use a loop to repeat one or more statements. The VOS language includes three types of loops: for, do...until, and while. The choice is mostly a matter of style and taste. Any given task that needs a loop can be written using any of the three types.

The italicized text in the following loop examples represents the following:

- *Statements*—One or more statements (which may themselves be loops) that are executed zero or more times as the loop repeats
- *Initialize*—An expression that is executed once before the for loop starts
- *Test*—A logical condition that is evaluated each time through the loop and determines whether the loop continues executing
- *Increment*—An expression that is executed once at the end of each iteration through the for-loop

For Loop

The *for* loop begins by testing for a condition that determines whether to execute the loop operation. If test is True, the loop executes until the increment goal is reached. If the test is False the loop doesn't execute, so if the test is False the first time through, the statements inside the loop are never executed and the increment is never executed.

The test may be left blank, in which case the value is always assumed to be True and the loop repeats forever, or until exited by means of a goto, jump, or return statement. You can leave out the initialize expression if it's not required.

```
for (initialize ; test ; increment)
   statements
endfor
```

While Loop

Like the for loop, the *while* loop also begins by testing for a condition that determines whether to execute the loop operation. If the test is True, the loop continues to run. If the test is False the loop stops executing so if the test is False the first time through, the statements inside the loop never execute. This loop differs from the *for* loop in that the condition is tested before each loop.

```
while ( test )
   statements
endwhile
```

Do...Until Loop

The *do...until* loop begins by executing the operation and then evaluates the condition that determines whether the operation needs to execute again. This type of loop continues until the test is True.

```
do
    statements
until ( test );
```

Example 6-8 Using loops

The most common example of a loop is stepping through all values of a variable from 1 to an upper limit, say Max. The following loops all compute the sum 1+2+...+Max using a variable called n:

```
Sum = 0;
for (n = 1; n <= Max; n++)
    Sum = Sum + n;
endfor


Sum = 0;
n = 1;
while (n <= Max)
    Sum = Sum + n;
    n++;
endwhile


Sum = 0;
n = 1;
do
    Sum = Sum + n;
    n++;
until (n > Max);
```

If you don’t like the short-hand n++, you can use n = n + 1 or n += 1 instead. Also, Sum = Sum + n could be replaced by Sum += n if you prefer.

A loop that repeats forever (or until exited by a goto, jump, or return command) may be written using a for-loop:

```
for (;;)
   # ...
endfor
```

—or a while-loop:

```
while (1)
   # ...
endwhile
```

Remember that True is represented as “1”, so the *while* test is always True.

Utilizing Functions

A *function* is a sequence of statements that has a name and produces a value. Functions can have one or more *arguments*, also called *parameters*. Arguments are values that are copied into the function. Inside the function, these arguments behave like variables and are accessed as normal by using their names. However, there is one difference between arguments and variables—an argument may not be assigned a new value.

There are three types of functions in VOS:

- Built-in functions—These functions are hard-coded into VOS and VLC, and are always available to be used in a program.
- User-defined functions—These are functions you write in VOS source code.
- RLL functions—These functions are written in C or C++. They are loaded from binary files called Runtime Link Libraries (RLL). In Windows, an RLL is a DLL.

The syntax for calling a function is the same for all three function types. The name of the function is given, followed by a list of arguments in parentheses, and separated by commas. Even if there are no arguments, the parentheses must still be given. Some examples of calls to functions are:

```
MediaPlayFile("Hello.vox");# Built-in function

MyFunc();# User-defined

code = ASUse(recordset);# RLL
```

Because the syntax is exactly the same, you can't tell from the call to the function whether it is built-in, user-defined, or contained in an RLL (although most built-in functions display as highlighted blue).

When a function name is used in an expression, the function is executed (this is known as calling the function) and the value produced by the function (called the return value) is obtained. The name of the function and its list of arguments are thrown away and replaced by the return value.

All functions return values. If no return value is specified, an empty string (a string containing zero characters is written as "") is returned. Many functions, such as trace, don't return any useful value—trace always returns an empty string. If the return value is not used in the statement, then VOS simply ignores the return value:

```
trace(2); # Return value not used
```

If a useful value is returned, it may be used in an expression. For example, if a function named MyFunc requires two arguments and returns a value, then the following are valid statements:

```
MyFunc(1, 2); # Ignore return value

x = MyFunc(1, 2); # Store return value in x

# Use return value in expression:

y = x*MyFunc(1, 2) + z;
```

The arguments to a function may be constants, variables, or expressions. The following are also all valid:

```
MyFunc(1, 2); # Constants

MyFunc(x, y);# Variables

MyFunc(z, y+123); # An expression
```

An expression used as a function argument may itself contain function calls. Like most other features of the VOS language, nesting or recursion to many levels is permitted.

Built-in Functions

VOS includes an extensive range of built-in functions for you to use. This list describes some of the most important groups of built-in functions. For a complete list, see the Graphical VOS online help.

Trunk—The Trunk functions control Trunk Resources, which let you make outbound calls, answer or reject incoming calls, and get information on incoming calls, such as Caller ID (ANI), DNIS, and Caller Name.

Media—The Media functions control Media Resources, which let you play and record sound files, get digits from the digit buffer, and play tones.

Fax—The Fax functions control Fax Resources, which support fax transmitting and receiving on a call previously established on a trunk.

Conference —The Conference functions control Conference Resources, which support conversations with three or more participants.

Database—The ADO RLL is the recommended way to add mission-critical database functionality to your VOS program. This RLL supports Microsoft ActiveX Data Objects, including extensive support for SQL, transaction processing, multi-user locking and other important features. The built-in db_ functions support database and index access using dBase-compatible DBF database files and Clipper-compatible NTX indexes. These are useful for small ad-hoc databases, especially read-only or rarely-updated configuration information.

Serial Communication—The ser_ functions support RS-232 serial communication through COM ports.

VOS Box—The vid_ and kb_ functions provide for simple screen output and user input to a VOS program. These functions are most useful when developing and debugging because they provide “quick and easy” ways to get data in and out of your program interactively.

User Interface—You may want to create a graphical user interface. You can do this by using the Dialog User Interface RLL or the NetHub RLL / ActiveX plus a tool such as Visual Basic, Delphi, Visual C++ or other Windows visual tool to create the GUI itself.

Task Management—There are several built-in functions for managing tasks, including spawn, chain, exec, arg, getpid ,and kill.

Inter-Task Communication—VOS provides four methods for communicating between tasks: global variables (glb_ functions), semaphores (sem_ functions), messages (msg_ functions), and groups (grp_ functions).

String Manipulation—There is an extensive set of functions for manipulating strings, including length, substr and many others.

File Access—The fil_ family provides functions for opening, reading, writing, seeking and locking files.

User-Defined Functions

You can define your own functions in the VOS language. Function definitions appear following the endprogram statement that ends the main program.

Let’s define a simple function named Add2 which adds two numbers (while this isn’t a particularly useful example, it clearly illustrates how to define a function):

```
func Add2(Arg1, Arg2)

   return Arg1 + Arg2;

endfunc
```

The function definition starts with func, is followed by the name of the function, and then by a list of zero or more parameter names separated by commas. The function definition ends with endfunc.

The return statement exits the function. There may be any number of return statements which may appear anywhere within the function.

A return statement may look like this:

```
return;
```

—in which case an empty string is returned or an expression may be given:

```
return expression;
```

Here, the expression is evaluated and the resulting value is returned. In our example, we want to add the two arguments Arg1 and Arg2, so Arg1+Arg2 is the expression we need.

If no return statement is given, an empty string is returned when execution reaches the endfunc statement.

Any series of statements that would be valid in the main program can be included in a function (except the label statement).

Optionally, a dec...enddec block may be included immediately following the func statement. This is used to declare variables and constants for use only within the current function.

Example 6-9 Adding a dec...enddec block after a function

```
func Add2(Arg1, Arg2)

   dec

       var Sum : 10;

   enddec


   Sum = Arg1 + Arg2;

   return Sum;

endfunc
```

Function arguments are passed by *value.* This means that a string value is copied into each argument name (Arg1 and Arg2 in our Add2 example) when the function begins execution. You can't assign a value to an argument:

```
Arg1 = 1; # Illegal!
```

This means that you can't change the value of a variable by passing the name of the variable as a function argument. This is because it is the value stored in the variable, not the variable name, that is copied into the function. If you need to

change variables passed into a function, you can use the *unary indirection* operators & and *. The expression &v is the internal VOS variable number of v, the expression *v is the variable whose internal number is v. These two can be used to pass variables *by reference* into a function. See the Graphical VOS online documentation for more details.

RLL Functions

VOS language functions may appear in separate files. If a function has not been defined in the main source code file, VLC searches for an external file containing the function. For example, if a function named MyFunc has not been defined, VLC searches for a file named MYFUNC.FUN. The first eight characters of the function name are converted to upper case, and the extension .FUN is appended. You can specify a search path for function files with the [FunFileDirs] section of the VOS Settings file.

Function libraries are created using the mkvl utility. Function libraries can be encrypted so that you can distribute library files without disclosing your source code. See the Graphical VOS online help for mkvl and dmpvl for more details.

This function libraries, called Runtime Link Libraries (RLLs) are extensions to the VOS language written in C or C++. An RLL contains one or more functions that can be called from a VOS application. A function in an RLL can be used exactly like a built-in function in VOS. The function may take either a fixed or variable number of arguments.

RLLs are specified in the VOS Settings file. Each RLL is loaded by VOS or VLC each time they are run. In a Windows environment, RLLs are implemented as DLLs (Dynamic Link Libraries).

To create your own RLLs you can use C/C++ along with the VOS C Developer's Toolkit and a Microsoft compiler.

 *Caution:*

If more than one RLL is used, VOS must load RLLs in the same order that VLC used when compiling the application. This requires that RLLs are specified in the same order in the Settings file used by VLC and VOS. Of course, using the same Settings file for both VOS and VLC is the easiest way to achieve this.

RLL functions can be logged like built-in functions by using the trace(8) function or by setting the Override and RLLs entries in the [Trace] section of the VOS Settings file.

Writing VOS Script 7

This chapter begins with a tutorial that teaches you how to create two programs using the VOS language. Next, it analyzes the code from one of the programs you create. Finally, it provides some commonly used telephony functions written in VOS. It contains the following sections:

- Starting the VOS Program Tutorial
- Examining the Echo Program
- Using Basic Telephony Functions

Starting the VOS Program Tutorial

Before beginning this tutorial, make sure Graphical VOS is installed on your machine correctly. You are also going to use SimPhone, which is automatically included in the CT ADE installation.

If you have questions or want more information on any of the VOS code you see during this tutorial, see the Graphical VOS online help. The online help also includes a description of each VOS function.

Creating a Project for Your Program

Complete the following steps to create a Graphical VOS project (projects manage the files needed to run your VOS script programs).

1. Start SimPhone and Graphical VOS from the Windows Start menu.
2. On the Welcome to Graphical VOS dialog, select Create a new project, then click OK.

   The New dialog displays.

Figure 7-1 Welcome to Graphical VOS dialog

3. On the New dialog:
   a. Make sure that Project is the selected File Type.
   b. In the File name field, name your project **LanguageTutorial**.
   c. In the Location field, type the following path for the project:

```
C:\Program Files\Parity Software\Graphical
VOS\Tutorial\Language
```

Figure 7-2 New dialog

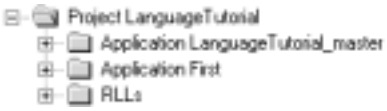
   d. Click OK, and Graphical VOS creates your project.

Adding a Program to the Project

Now it's time to add a VOS program to the project.

1. From the main menu in Graphical VOS, click File | New.
2. In the New dialog:
   a. Make sure that VOS Program is the selected File Type.
   b. In the File name field, name the program **First**.
   c. Make sure the Add to Project checkbox is checked.
   d. Click OK to create the new program.

   A folder called Application First is added to the project tree in the left window:



3. Double-click the Application First folder, and you'll see an icon labeled First.vs in the project tree.

The Editor window to the right is blank because this new file currently contains no text.

4. Type the following program into the Editor window:

```
program

    vid_write("I am a VOS program!");

    vid_write("Type any key to exit...");

    kb_get();

    vid_write("Exiting now.");

    exit(0);

endprogram
```

Compiling and Running the First Program

When you compile a VOS program, it gets converted from a form that you can understand (such as a source file, like First.vs) to a form that VOS can execute (such as a binary file—also called an executable file).

1. From the Graphical VOS main menu, choose Run | Compile First.vs.

   If you didn't make any typing errors, you'll see a message telling you that your program has been compiled:

```
Compiling...
first.vs
Compilation Complete
```

   The VOS Language Compile (VLC) created a binary file called First.vx. You can use Windows Explorer to see that the binary file has been created. If you see error messages from VLC, check your typing by carefully comparing the file you created with the source code provided.

   Next, run the First program:

2. From the main menu, choose Run | Run First.vs.

   Notice that Graphical VOS compiled your program again. Any time you run a program in Graphical VOS, it automatically compiles the program first. After compiling the program, Graphical VOS starts the VOS runtime engine. You can see the stopwatch icon ticking in the Windows system tray, and a new window called VOS Box opens (if you have previously disabled the VOS box, it doesn't open).

3. Right-click the VOS Box icon in the Windows system tray, and choose Restore.

The following message appears in the VOS Box:

```
I am a VOS program!
Type any key to exit...
```

4. Type any key, and the VOS Box window closes.

   If you didn't see that message, try looking at the VOS log file. From the main menu, choose View | Runtime Log. The VOS1.LOG file is created automatically by VOS. It displays a new message each time VOS is started, and each time a condition arises that may indicate an error in a program or hardware problem. If you don't find a message indicating the source of the trouble, it's time for your first tech support call. For more information, see the chapter titled "Technical Support" in this guide.

Reading the Source Code

You should find the source code to be mostly self-explanatory. For example, the word *program* starts a program, and the word *endprogram* marks the end of the program.

The vid_write function asks VOS to write the given string to the VOS Box. The kb_get function waits for a keystroke, and returns the character pressed, although we ignore the returned value in this example. Most VOS programmers use languages like Visual Basic or Visual C++ to develop interfaces for VOS programs so you probably won't use functions like vid_write and kb_get very often, but they're appropriately simple for an example like this.

The exit function stops the VOS runtime engine with the given exit code (which can be tested by the IF ERRORLEVEL command in batch files, etc.).

For more information about interpreting the source code, see the chapter titled, "VOS Language Concepts" in this guide or the Graphical VOS online help.

Creating a Second Call Processing Program

Now you're ready to add another VOS program to the project.

1. From the main menu in Graphical VOS, click File | New.
2. In the New dialog:
   a. Make sure that VOS Program is the selected File Type.
   b. In the File name field, name the program **Echo**.
   c. Make sure the Add to Project checkbox is checked.

Figure 7-3 New dialog displays when you choose New from the Graphical VOS he File menu

d. Click OK to create the program file.

3. Double-click the Echo icon in the project tree, and then type the following code in the Editor window:

```
program
    TrunkWaitCall();
    TrunkAnswerCall();
    MediaRecordFile("msg.vox", 15);
    MediaPlayFile("msg.vox");
    TrunkDisconnect();
    restart;
endprogram
```

Compiling and Running the Second Program

Complete the following steps:

1. From the main menu in Graphical VOS, choose Run | Run Echo.vs.

   The program automatically compiles first and then runs. Again, you can see the stopwatch icon and the VOS Box icon in your system tray.

   The program you wrote executes, and then waits for a call to come in.

2. Click Ring in the SimPhone main window.

   You'll see the Line State icon in SimPhone go off-hook, which means that your application has answered the call.

3. Say something into your sound card's microphone and then click any SimPhone digit button.

   This stops the recording, and the message is played back to you. As soon as the message playback has finished, the program ends the call and starts at the beginning. You can then call in again if you want.

4. When you are finished, stop the program by choosing Run | Stop Running from the Graphical VOS main menu.

This completes the tutorial. In the next section you analyze the code in the Echo Program.

Examining the Second Program

In this section we'll dissect the Echo program you created. The following table offers a brief explanation of what happens at each line. The next section provides a more in-depth explanation.

Command	Action
`program`	Start here
`  TrunkWaitCall();`	Wait for call
`  TrunkAnswerCall();`	Answer call
`  MediaRecordFile("msg.vox", 15);`	Record sound to file
`  MediaPlayFile("msg.vox");`	Play back sound
`  TrunkDisconnect();`	Terminate call
`  restart;`	Go back to first statement
`endprogram`	End of main program

Echo Program Code Detail

Every program must begin with the word *program*, and end with the word *endprogram*.

TrunkWaitCall Command

Command: `TrunkWaitCall();`

Action: Waits for an incoming call on the current trunk (phone line)

Description: This command tells VOS to suspend the program until an incoming call is received on the given line. When the call comes in, VOS "wakes up" the program, which then continues execution.

TrunkAnswerCall Command

Command: `TrunkAnswerCall();`

Action: Answers the call on the current trunk line

Description: This command instructs the Dialogic board to "go off hook," meaning to answer the phone. This action corresponds to picking up the handset on a telephone.

MediaRecordFile Command

Command: `MediaRecordFile("msg.vox", 15);`

Action: Records from the current line to a hard disk file named msg.vox

Description: This command turns the Dialogic board into a digital tape recorder. The sound received on the phone line is digitized and saved to a file on the computer's hard disk. Just as with MediaPlayFile, there is one mandatory parameter to this command, the hard disk file name to store the digitized sound. In addition to the file name, by setting the second parameter, you can control the maximum length of the recording, in seconds. The example shows MaxSecs set to 15, so the recording can be up to 15 seconds long.

MediaPlayFile command

Command: `MediaPlayFile("msg.vox");`

Action: Plays back sound from msg.vox on the current line

Description: This command plays back sound from a hard disk file. It has one mandatory parameter, which is the name of the file.

TrunkDisconnect command

Command: `TrunkDisconnect();`

Action: Terminates call on the current line

Description: This command finishes the call by putting the line back "on-hook" to hang up.

Restart Command

Command: `restart;`

Action: Goes back to the beginning of the program

Description: This command goes back to the beginning and starts executing again. You can think of it as jumping back to the word *program* and resuming execution from the first command.

Using Basic Telephony Functions

The next two tables list some of the main capabilities of the telephony boards. The following section describes how to write several functions you're most likely to use.

Audio Processing

This table lists audio processing commands:

Action	Command
Play speech from file	`MediaPlayFile`
Record speech to file	`MediaRecordFile`
Get touch tones from caller	`MediaGetDigitBuffer`

Telephone Signaling

This table lists telephone signaling commands:

Action	Command
Wait for incoming call	`TrunkWaitCall`
Answer incoming call	`TrunkAnswerCall`
Terminate a call	`TrunkDisconnect`
Make outbound call	`TrunkMakeCall`
Detect caller hangup	`onhangup`

Programming Basic Features

This section shows you the code necessary to program basic computer telephony features using VOS script.

Answering an Incoming Call

VOS answers a typical call using the following command sequence:

Action	Command
1. Wait for ring	`TrunkWaitCall();`
2. Go off-hook	`TrunkAnswerCall();`
3. Play greeting	`MediaPlayFile(filename);`

For more information and an example program that answers inbound calls, see the Graphical VOS online help.

Making an Outgoing Call

VOS usually makes an outgoing call using this command sequence:

Action	Command
1. Make outbound call	`TrunkMakeCall();`
2. Check new state	`State = TrunkGetState();`
	`Glare = TrunkGlare();`

The new state of the trunk should be Connected, NoConnect, or InboundRinging. Connected means that the call went through or that a condition called *glare* occurred, that is, an inbound call arrived just as you were attempting an outbound call. You can test for glare with the TrunkGlare function. A NoConnect state means that the call was not completed (for example, there was a busy tone, or the line rang off the hook without being picked up). InboundRinging also indicates a glare condition.

For more information and an example program that makes outbound calls, see the Graphical VOS online help.

Terminating a Call

To end a call, simply use the TrunkDisconnect function. Most programs are designed to start again at the beginning and then prepare to receive another call. You can do this using the restart command:

Action	Command
1. Hang up	`TrunkDisconnect();`
2. Return to start of program	`restart;`

Detecting a Disconnect

The caller may hang up at any point in the VOS program. Consequently, the caller might hang up when the program is not currently waiting for a hang-up or when it's executing an action.

The most commonly used mechanism in VOS is the onhangup function. When a disconnect is detected, VOS interrupts the program wherever it is currently executing, and jumps to the onhangup function. The onhangup function can contain any VOS commands. The onhangup function usually finishes with either a return command or a restart command. A return command sends the program back to the point where it was originally interrupted, and continues exactly as

before (except that variables may be changed in the onhangup function). If a restart command is executed, the program goes back to the start of the program and is ready to receive the next call.

Example 7-1 VOS code that creates an onhangup function

```
onhangup

   TrunkDisconnect();

   restart;

endonhangup
```

Suitable for most VOS applications, this code reacts to a caller hang-up by disconnecting and going back to the beginning of the program, which presumably either waits for the next call to arrive or initiates a new call.

A problem with using onhangup can arise if the program is in the middle of doing something important when the disconnect signal arrives. For example, it may be making updates to a database. This situation can be handled using the TrunkDeferOnHangup and TrunkClearDeferOnHangup commands, which allow a disconnect signal to be *deferred*. Calling TrunkDeferOnHangup says, "I am beginning to do something important—don't jump to onhangup even if a disconnect signal does arrive." Calling TrunkClearDeferOnHangup says, "OK, if a disconnect signal did arrive, now jump to onhangup, otherwise just carry on as usual."

We can add disconnect processing to our example program by adding an onhangup function.

Example 7-2 VOS code that adds an onhangup function

```
program

      voslog("Program started");

      TrunkWaitCall();

      TrunkAnswerCall();

      sleep(10);

      MediaRecordFile("msg.vox", 15);

      MediaPlayFile("msg.vox");

      TrunkDisconnect();

      restart;

endprogram

   onhangup
```

```
      TrunkDisconnect();

      restart;

   endonhangup
```

Creating a Touch Tone Menu

One of the basic tools utilized in automatic call processing is the ability to play a menu and get a response from the caller. A typical audible menu pattern reads like this:

"*Please press 1 to do this, press 2 to do that, press 3 to do...*"

In order for the menu to work correctly, the pre-recorded file containing the voice saying the menu must play first. Then VOS must wait for the caller to enter a touch-tone digit.

Example 7-3 VOS code that creates a touch tone menu

```
 MediaPlayFile("menu.vox");

 MediaWaitDigits(1);

digit = MediaGetDigitBuffer();

switch (digit)

case 1:

   do_one();

case 2:

   do_two();

case 3:

   do_three();

default:

   bad_digit();

endswitch
```

The MediaWaitDigits function waits for the caller to enter one or more digits. The function has one mandatory parameter, that is, the number of digits to get. Thus, MediaWaitDigits(1) waits for a single digit and moves it to the digit buffer (the place where VOS stores digits dialed by the caller).

The MediaGetDigitBuffer command returns the contents of the digit buffer.

In this example we used the commands: do_one, do_two, do_three, and bad_digit. These are *user-defined functions*, that is, new commands you can invent by writing VOS script. User-defined functions are called functions or subroutines in other languages.

Debugging in Graphical VOS 8

This chapter provides a tutorial in which you create and debug an application. It contains the following sections:

- Creating an Application
- Debugging the Application

Note: Debugger is for use with Graphical VOS only.

Creating an Application

In this section, you use Graphical VOS to create an application. In the next section you debug your new application using CT ADE Debugger and SimPhone.

Before You Start

Before beginning this tutorial, make sure CT ADE with Graphical VOS 8.0 or higher is installed correctly. You can run this tutorial in either evaluation or development mode. You don't need a telephony card for this tutorial because you're going to use SimPhone.

If you have questions or want more information on any of the VOS code you see during this tutorial, see the Graphical VOS online help, which includes a description of each VOS function.

Creating a new Project

Complete the following steps:

1. Choose one of the following:
    a. If you don't have Graphical VOS running, launch it from the Start menu and select the Create a new project button on the Welcome to Graphical VOS dialog.
    b. If Graphical VOS is running, close any open projects and choose New from the File menu.
2. In the New dialog:
    a. Select Project for the File Type.
    b. Name the project **Debugger**.

c. Save it in the following directory:

```
C:\Program Files\Parity Software\Graphical
VOS\Tutorial\Debugger directory
```

Figure 8-1 New dialog displays when you choose New from the Graphical VOS File menu

d. Click OK to create the project.

Now add a new VOS Program to the project:

3. From the File menu, choose New and then complete the following:

   a. In the New dialog under File Type, select VOS Program.

   b. Name the file **inbound**.

   c. Make sure the Add to Project box is checked.

   d. Click OK to create the file to add it to your Graphical VOS project.

In the Project window, you can see a tree that shows the project's files: a master application, an RLL, and the inbound.vs application:

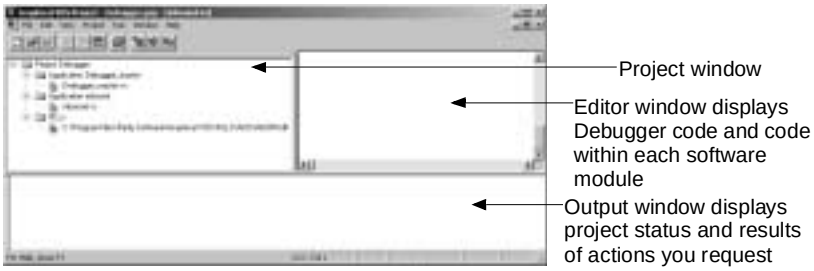


Figure 8-2 Graphical VOS Project window

4. Double-click the inbound.vs icon in this tree, and you can see the empty file in the Editor window to the right.

   The window on the bottom is the output window.

5. Click in the Editor window on the right and type the following code into it:

```
program
  TrunkWaitCall();
  TrunkAnswerCall();
```

```
  if (TrunkGetState() strneq "Connected")
     voslog("Unexpected trunk state ", TrunkGetState());
     stop;
  endif
  InboundCall();
  TrunkDisconnect();
  restart;
endprogram


onhangup
  if (TrunkGetState() streq "RemoteDisconnected")
     TrunkDisconnect();
  endif
  restart;
endonhangup

func InboundCall()
  MediaPlayFile("..\Prompts\HelloInbound.vox");
endfunc
```

6. Choose File | Save.

Debugging the Application

Now you're ready to debug the application you created.

Starting Debugger

Complete the following steps:

1. In the project window, right-click the Application inbound folder and choose Debug Application.

   Graphical VOS starts Debugger and VOS. The Debugger displays in the right window—which is now called the Tasks window, and VOS displays the VOS Box dialog.

2. Minimize the VOS Box dialog.

3. If you need to, right-click on the Programs icon in the Tasks window and choose Expand Tasks Tree so that you can see Task 0.

Stepping through Code

Now we can look at the Inbound program more closely.

1. In the Tasks Window, double-click Task 0 to see the source files for the task. Then double-click inbound.vs to switch to the Source File window and see the source code.

Figure 8-3 Source code for the inbound.vs task

The next statement to be executed in the task is highlighted and displayed with a “blue arrow” icon. Because we haven’t executed any lines of code yet, the next statement to execute is the first statement in the program:

```
TrunkWaitCall();
```

This waits for an incoming call. The program waits indefinitely until a call arrives.

2. Click the Step Next icon, and the highlight and blue arrow disappear. If you do not see this icon in the toolbar, choose View | Debugger Toolbar.

   VOS is executing the first line. In order for the function to return, the application must receive an incoming call.

3. On the SimPhone main window, click Ring to simulate a call to the voice card.

   Now the highlight and blue arrow move to the second line in the program, which is:

   ```
   TrunkAnswerCall();
   ```

4. Click Step Next again and VOS answers the call.

   The blue arrow and highlight move to the next line in the program.

Next we are going to execute a few lines of code without having to click Step Next repeatedly.

Using Breakpoints

A breakpoint is a condition that causes a VOS task to stop execution and enter a Paused state. First let's add a breakpoint to a line:

1. Click once on the line of code that reads:

   ```
   InboundCall();
   ```

   A different highlight, called the cursor, appears. By default, the cursor is bright green.

2. In the toolbar click the Toggle Breakpoint icon and a red hand appears next to the line.

Figure 8-4 The red hand specifies the point at which the Debugger stopped

3. Next, click the Resume icon in the toolbar and the program executes up to the breakpoint.

Viewing the Call Stack

The next line to execute in the main program invokes a user-defined function:

```
InboundCall();
```

To step through this function:

1. Click the Step Into icon.

   Clicking this icon executes up to the first line of the function.

2. Next, click the Call Stack icon to display the Call Stack dialog.

Figure 8-5 Call Stack Dialog

The function call stack is a display showing the hierarchy of function calls that the program went through to get to its current position. You can see that the task is currently paused on line 21 of inbound.vs, in a function called InboundCall. That function was called from line 8 in the main program in inbound.vs.

3. Click OK to close the Call Stack window.
4. Click the Step Next icon to execute the next function, playing the HelloInbound.vox file.

   Finally, close the Debugger.

5. From the Run menu, click Stop Debugging.

For more information about the Debugger, see the Graphical VOS online help.

Troubleshooting 9

This chapter describes how to fix or avoid common problems you may encounter with your Graphical VOS or CallSuite application. It contains the following sections:

- Overview
- Troubleshooting in Graphical VOS
- Troubleshooting in CallSuite

Overview

There are six general steps you need to take to troubleshoot any problem you are experiencing:

1. Find out how to consistently reproduce the symptom.

   If a problem cannot be reproduced, it is much harder to analyze.

2. Localize the issue to the Graphical VOS function/CallSuite method or property setting that is causing the symptom.

3. Examine the information contained in the log file to determine which function/method or property relates to the symptom you are seeing, as well as the accompanying messages.

   The problem is almost always caused by a single function/method or property setting.

4. Check the latest information from your support provider regarding known bugs to see if there is anything related to your symptom.

5. Simplify your test project as far as possible.

   For example, you might run the project on only one telephone line or remove all code unrelated to the symptom you are seeing, such as database accesses.

6. Contact Technical Support via e-mail, fax, or phone to report the problem.

The next two sections describe troubleshooting in more detail in both Graphical VOS and CallSuite.

For more information about reaching Technical Support, see the chapter titled, “Technical Support” in this guide.

Troubleshooting in Graphical VOS

At runtime, VOS creates a file named VOS1.LOG that reports information about how your application is running that you may find useful when you are debugging and troubleshooting. This file is created in the same directory where your program resides, but you can specify a different directory if you want. For details on how to do this, see the section titled, "Configuring the VOS1.LOG File" in this chapter.

Viewing the Log File

You can access the log file whether or not you are in the IDE (Integrated Development Environment). To view the log without launching Graphical VOS, open any text editor, browse to find the VOS1.LOG file in the application's project folder, and then open it. To view the runtime log from within the IDE, open the application in Graphical VOS, and then choose View | Runtime Log.

Figure 9-1 VOS1.LOG file

When VOS1.LOG reaches the maximum size, 256 Kb, VOS renames it VOS2.LOG (deleting any existing file with that name), and starts a new VOS1.LOG file. Therefore, VOS1.LOG always contains the latest information, and VOS2.LOG (if present) contains entries up to the start of VOS1.LOG. Distributing the information in this way allows you to control the size of the log file so that it doesn't consume too much space on your system.

By default, VOS appends to any existing VOS1.LOG file, and the old file is not deleted when VOS starts. To change this behavior, see the section titled, "Configuring the VOS1.LOG File" in this chapter.

While it may seem confusing at first, learning to use the log file can save you hours of troubleshooting on systems under development and on systems installed at customer sites. We recommend that you read this chapter carefully and take the time to review your log files regularly. The effort will repay itself many times over.

Interpreting the VOS1.LOG Contents

This section explains the entries in the VOS1.LOG file.

VOS Version

When VOS starts, an entry like the following is written to the log file:

```
000424 121445.92 VOS 7 (Debug) Built Apr 19 2000 14:23:51
```

The first field is a time-stamp and it has this format: YYMMDD HHMMSS.CC (CC is hundredths of a second).

```
000424 121445.92
```

Most lines in the log file have a time-stamp showing when the line was written.

The second field shows the version of VOS—version 7 in this case, and that it is running in Debug mode.

```
VOS 7 (Debug)
```

 Tip:

In Debug mode, VOS produces more detailed and complete information in the log file. We recommend that you use Debug mode when creating your application and even when installing live systems, unless the slower performance or greater memory use of Debug mode creates problems.

The last field reflects the date and time that VOS was built.

```
Built Apr 19 2000 14:23:51
```

Seeing when VOS was built provides more exact information than the version number because the build time distinguishes between minor versions which are shipped with bug fixes and enhancements.

VOS Exit

When VOS terminates for any reason, two entries are written to the log file giving the reason why VOS is stopping and confirming that it did stop. Typical entries look like this:

```
000424 121511.11 Stopping: Command from external application

000424 121511.21 VOS stopped.
```

Unless VOS crashes catastrophically, these entries are always present after VOS terminates.

Using Built-In Function Calls

The log file entry for a typical built-in function call looks like this:

```
000426 140726.13 inbound:00c3[0] @B time() = 140726
```

The fields following the time stamp (inbound:00c3[0]) identify the .vx file, code address, and task number that made the built-in function call. In this case, the call was made from inbound.vx at code address 00c3—which is task number 0. The code address is given in hex, and this is one of the few cases where information is written to the log file in hex rather than in decimal notation. The code address can be converted back to a source file name and line number using the dmpvx utility.

The @B code identifies the type of information being logged. There are a number of different @ codes, for example, @B is a built-in function and @R is an RLL function. You can use these codes in an editor or searching type of filter to quickly extract the information of interest when examining a log file. For more information, see the section titled, "Interpreting @ Codes in the Log File" in this chapter.

The time() = 140726 entry shows the name of the built-in function, all the arguments passed to the function (double-quotes are not used around the string values), and the returned value from the function.

Entries display after the command has executed, but you can configure the log file to display the command before execution if you want. To do this, see the section titled, "Configuring the VOS1.LOG File" in this chapter.

Blocking Function Calls

Some built-in functions may block the task, in other words they do not return immediately. Instead the task is "put to sleep" (suspended) while other tasks are allowed to continue. A simple example is the sleep function, which suspends the task for a given number of tenths of a second.

VOS cannot show the returned value from a blocking function at the time the function is called so the log shows two lines: one where the function is called, and a line later on showing when the task resumes and the return value being returned at that point. This example shows how a sleep function might appear in the log file:

```
000426 144512.58 inbound:00cf[0] @B sleep(5)

000426 144513.08 ResumeTask(0, )
```

No return value is shown when the @B line is generated, since it is not yet known. The ResumeTask line shows where the task is restarted. (The format for this line is: ResumeTask(TaskNumber, ReturnedValue).)

The sleep function returns an empty string, which you can see in the ResumeTask log entry. The TaskNumber is needed to match the function call to the ResumeTask. You can see that sleep was called by task number 0. Note that there may be several lines between the sleep and ResumeTask line if other tasks are running.

You can also see from the time stamps that sleep did indeed suspend the task for 13.08–12.58 = 5 tenths of a second.

Blocking Functions and Topaz State Changes

By default, VOS waits until one function requiring a Topaz Resource completes before executing the next function.

Example 9-1 Recognizing blocking in log entries

The following log file entries were produced by invoking the MediaPlayFile function:

```
000426 145231.72 inbound:00c6[0] @B MediaPlayFile(hi.vox) = 1

000426 145233.54 [0] @Z Media:0 Playing->Idle

000426 145233.54 ResumeTask(0, 1)
```

In this example, Media Resource 0 is being blocked by this function in VOS task 0: @B MediaPlayFile (hi.vox) = 1, and the VOS built-in function that is blocking is @Z Media:0 Playing->Idle.

An @Z entry shows that a Topaz Resource State is changing. In this case Media Resource 0 is changing from the Playing state to the Idle state. (For more on Topaz Resource states, see the Graphical VOS online help.)

The ResumeTask(0, 1) entry shows that the task has been “woken up” again. In this example, task 0 was resumed and the return value from MediaPlayFile is 1.

Asynchronous Mode

Asynchronous mode lets you start one function that requires a Topaz Resource and then continues your program’s execution before that function finishes. When you’re running in asynchronous mode, functions that don’t use the Topaz Resource can appear between the function that blocked the Resource and the notification that the Resource was blocked.

Example 9-2 Recognizing blocking in log entries while in asynchronous mode

Time() function executes while the MediaPlayFile function is playing:

```
000426 152447.56 inbound:00b2[0] @B MediaEnableAsync() = 1

000426 152447.59 inbound:00ce[0] @B MediaPlayFile(hi.vox) = 1

000426 152447.59 inbound:00d5[0] @B time() = 152447
```

```
000426 152447.59 inbound:00df[0] @B MediaWait() = 1

000426 152449.40 [0] @Z Media:0 Playing->Idle

000426 152449.40 ResumeTask(0, 1)
```

Interpreting @ Codes in the Log File

The following codes are used as prefixes to various log file entries. These codes are designed to allow a text editor or text search program to identify relevant lines in VOS1.LOG.

Code	Meaning
@b	Built-in function call and the arguments used before the call is made. Tracing before and after function calls must be enabled to see @b codes
@B	Built-in function call and result. At this point, the call has already been made
@D	Dialogic API call or event
@E	User error (voslog call starting with "@E...)
@F	Built-in function call failure message
@I	Invoke ActiveX object
@IG	ActiveX property get
@IM	ActiveX method call
@IP	ActiveX property put
@L	User log message to file only (voslog "@L...)
@N	Topaz Notify (a resource type- or Technology-specific notification, such as a digit detected event)
@O	Dialogic API call "on entry," before calling the actual function (API)
@Q	Topaz Operation Complete (completion event for a blocking Get/Set or [Un]Listen)
@r	RLL function call and the arguments used before the call is made. Tracing before and after function calls must be enabled to see @r codes
@R	RLL function when function returns
@S	Data Structure. This code dumps a C struct or similar passed in or out as a function argument
@V	Assignment to variable (watchvar)
@W	Built-in function warning
@X	Failed Dialogic API call or event
@Y	Technology-specific event
@Z	Topaz state change

Configuring the VOS1.LOG File

You can determine the output of the log file using two dialogs from the VOSEdit dialog list: the Logging dialog and the Tracing dialog.

1. Launch Graphical VOS and open a project.
2. Choose Project | VOS Settings.

   The VOSEdit dialog displays:

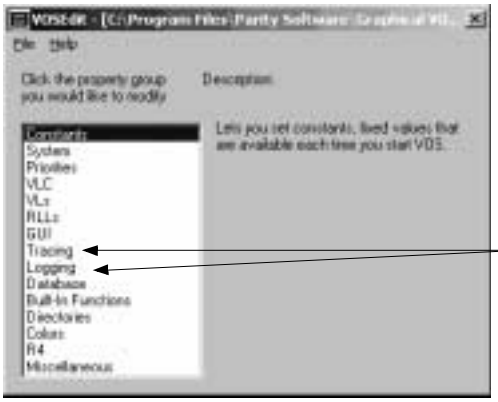


Figure 9-2 VOSEdit dialog

3. Choose one of the following:
   - — To access the Logging dialog, see the section titled, “Configuring the Logging Dialog” in this chapter.
   - — To access the Tracing dialog, see the section titled, “Configuring the Tracing Dialog” in this chapter.

Configuring the Logging Dialog

1. From the VOSEdit dialog, double-click Logging from the list.

Figure 9-3 Logging dialog

2. From this dialog, change the following log file settings as appropriate:

Entry	Meaning
Log file directory	Directory in which the VOSX.LOG files are stored
Maximum log file size (KB)	Sets the maximum size of the VOS log. Default is 256 Kb
Buffer output	Boolean. Set to 1 to enable buffering or 0 to disable buffering. If Buffered, VOS keeps > 1 line of log output in memory and writes a group of lines at one time. If not buffered, VOS writes each line as it comes
Append to existing log	Set to 1 to append new entries or to 0 to overwrite any old log when you start VOS

3. When you're finished, click OK to close the dialog.

   Your changes go into effect immediately.

Configuring the Tracing Dialog

1. From the VOSEdit dialog, double-click Tracing from the list.

   The Tracing dialog displays.

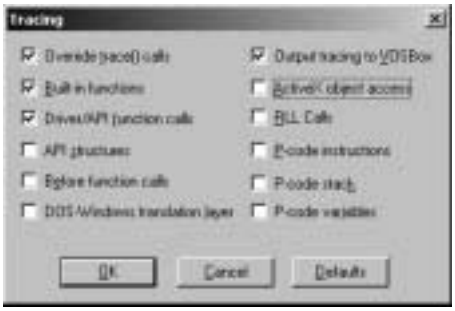


Figure 9-4 Tracing dialog

2. From this dialog, change the following trace settings as appropriate:

Entry	Meaning
ActiveX	Set to 1 to enable VOS to provide detailed tracing of object accesses (available in Debug mode only1), or set to 0 to disable it
Builtins	Set to 1 to trace all VOS built-in functions, or set to 0 to disable tracing of built-ins
Drivers	Trace API function calls
InOut	Trace both before and after function calls. This is useful when a function call hard-crashes VOS. Normally functions are logged only after they return, but if the function crashes this doesn't show in the log.

Layer	Log DOS-to-Windows translation layer for 'legacy' functions sc_, DTI_ etc.
OutputToVosBox	Set to 1 to display the tracing information in the VOS Box and write it to the log file, or set to 0 to just write it to the log file
Override	Set to 1 to override tracing options set with the trace() function in individual programs, or set to 0 to use the program's settings
Pcode	Set to 1 to trace all p-code instructions or set to 0 to disable p-code tracing (P-code tracing creates a large amount of logging)
RLLs	Set to 1 to trace all RLL calls or set to 0 to disable tracing of RLLs
Stack	Set to 1 to include the stack in p-code tracing or set to 0 to omit it
Structs	Set to 1 to trace API structure members or set to 0 to omit structures
Vars	Set to 1 to include variables in p-code tracing or set to 0 to omit them

3. When you're finished modifying the log file settings, click OK to close this dialog.

   Your changes go into effect immediately.

Troubleshooting in CallSuite

CallSuite components have the ability to keep a log of activities on the telephony channel. This log file, called VBocx.log, can help you understand how the CallSuite application is running as well as help you troubleshoot problems you encounter. When your application behaves in an unexpected way, search VBocx.log file for any errors, warning messages, events, or other information that may help you pinpoint the problem.

When you run an application in CallSuite, the VBocx.log file is created here:

```
C:\Program Files\Parity Software\Common\Log
```

Figure 9-5 VBocx.log file

When VBocx.log reaches the maximum size—250Kb, it is renamed VBocx2.log (deleting any existing file with that name), and a new VBox.log file is started. This prohibits the log file from growing without limits and filling up your disk. This also means that VBocx.log always contains the latest information, and VBocx2.log (if present) contains logs up to the start of VBocx.log. By default, CallSuite appends to any existing VBocx. file, and the old file is not deleted when CallSuite starts.

When you first see the CallSuite log file, it may look difficult to understand. However, we encourage you to review the log often and use the log entry descriptions provided here (and in the CallSuite online help) to become more familiar with it. As you become more experienced, you will find that it is a valuable resource.

Interpreting the VBocx.log file Contents

Depending on your settings via the SetLogLevel methods and INI file entries, the VBocx.log file contains useful records of what is happening as your application executes. Typical entries include:

- Methods and method parameters
- Telephony driver function calls
- Telephony events and messages
- Information about the runtime environment, such as the versions of DLLs used by the CallSuite components

There are two levels of logging: standard and detailed. Many entries appear in both standard and detailed although detailed logging contains additional entries.

Each line in the VBocx.log file has the following format:

```
HHMMSS.hh PpppTttt X[#] @c Text
```

The following log entries appear in both standard and detailed logging:

Log Entry	Description
`HHMMSS.hh`	Time in Hours, Minutes, Seconds and hundredths of a second when the log entry was made
`ppp`	Process index of the process that made the log entry
`ttt`	Thread index of the thread that made the log entry
	Process index and Thread index are zero-based counters that indicate which thread or process made the log entry. For example, in VoiceBocx, if you are running 24 lines in 24 .EXEs there will be 24 processes, each with 2 threads: one thread for the main application and one thread for the Dialogic drivers. The Process indexes will be numbered from 0 to 23 and the Thread indexes will be from 0 to 47.

X	CallSuite component that made the log entry and is one of the following: V = VoiceBocx, C = ChatterBocx, S = ConfBocx, F = FaxBocx, M = MatchBocx
#	Index number of Resource that made the log entry. If a Resource has not yet been assigned to the component (for example, during startup), the memory address of the CallSuite component displays in hexadecimal format.
@c	Code that indicates the log entry type. If the @c is not present, the entry is informational.
blank	The text is informational
@E	An error has occurred. Logged for all levels of logging (even if logging is disabled)
@L	User logging—your application has called the LogWrite method
@M	The CallSuite component is exiting a method
@N	A Topaz notify event has occurred
@P	The CallSuite component is exiting a property get or set
@Q	A Topaz Operation has completed
@V	An ActiveX event (often a SimPhone event) has occurred
@W	A CallSuite component has loaded
@Y	A Topaz event has occurred
@Z	A Topaz Resource's state has changed

These entries appear in detailed logging only:

Log Entry	Description
@m	The CallSuite component is entering a method
@p set	The CallSuite component is entering a property set
@P get	The CallSuite component is entering a property get

Each time an instance of a CallSuite component loads, you can see a confirmation of the path name to the control that loaded:

```
14:48:14.29 V[0x3cf6d70] @W VoiceBocx loaded from C:\Program
Files\Parity Software\CallSuite\Bin\VoiceBocx.dll
```

The exact date and time of the control build is useful information to Technical Support. It can be used to determine if known bugs or corrections made through point releases are present in the version of the control that you are using:

```
14:05:44.88 V[0x3cf6d70] ModuleRunModeInit of VoiceBocx: Topaz
Built May 18 2001 09:40:25
```

Methods are logged, together with their parameters:

```
14:05:48.82 V[i01] @M PlayFile FileName="hello.vox"
SoundFileType=0 StopTones="0123456789#*" SkipSecs=0
LengthSecs=0
```

Controlling Log Output

There are a number of ways you can modify the output that VoiceBocx sends to the log file. The primary approach is to use one of these SetLogLevel methods along with a log value that determines the degree of information you see for a particular CallSuite component.

SetLogLevel Methods	Description
`Call VoiceBocx1.SetLogLevel(LogLevel)`	Sets the current log level for VoiceBocx
`Call ChatterBocx1.SetLogLevel(LogLevel)`	Sets the current log level for ChatterBocx
`Call ConfBocx1.SetLogLevel(LogLevel)`	Sets the current log level for ConfBocx
`Call FaxBocx1.SetLogLevel(LogLevel)`	Sets the current level for FaxBocx
`Call MatchBocx1.SetLogLevel(LogLevel)`	Sets the current level for MatchBocx

Use one of the preceding methods along with one of the following three log levels to set the level of detail you want the log file to show for that component. For example:

```
Call MatchBocx1.SetLogLevel(LOG_Enabled)
```

Log Value	Value	Meaning
LOG_Disabled	0	No log output
LOG_Enabled	1	Normal log output (default)
LOG_Detailed	2	Maximum log output (required for logs sent to Technical Support)

Additionally, you can further alter log output by modifying the settings in the [Control] section of the VBocx.INI file, which is located in C:\WINNT.

Log File Setting	Valid Values	Default	Description
`HomeDir`	A valid DOS/ Windows drive and/or directory name.	C:\Program Files\Parity Software\Call Suite\Wizards	Used by CS Wizard. Do not change this path unless you also move everything residing in this directory to the newly specified location.
`LogDir`	A valid DOS/ Windows drive and/or directory name.	Write log file in current working directory.	Writes to a directory other than your application directory
`LogForce`	Yes, No	No	Forces detailed logging when LogForce is set to Yes even if the Log level is set to LOG_Disabled. This is convenient for troubleshooting at a customer site where you have a problem report. You can obtain log output without having to modify your application.
`LogStartup`	Yes, No	No	Logs parameters which affect the initialization process if set to Yes. For example, the Dialogic API calls made to initialize the telephony card together with .INI file entries are logged.
`LogBuffer`	Yes, No	Yes	Not currently used
`ErrorPopup`	Yes, No	Yes	Displays an error message when CallSuite receives an error. Setting this parameter to Yes stops application, so you may find it more useful during debugging than at production time
`LogFileAppend`	Yes, No	No	Appends to an existing log file when VoiceBocx is started if set to Yes. If set to No, VoiceBocx deletes any existing log file from previous runs before starting.

Log File Setting	Valid Values	Default	Description
`LogErrorPopup`	Yes, No	Yes	Displays an error message when CallSuite experiences a problem writing to the log file. This is useful for debugging, but may not be convenient during production time because it stops the application.
`MaxLogFileSize`	1024 to 2000000000	100000 (100 Kb)	Specifies the maximum size of the log file. If the file becomes larger than what you specified, then the current log file is renamed VBocx2.Log—over-writing any existing file with this name, and a new log file is started.

Technical Support 10

This chapter describes what steps to take before you contact Technical Support to ensure fast resolution to your problem.

- Overview
- What We Need From You
- Contacting Us

Overview

You can send your support issue to Technical Support via e-mail, fax, or phone. We strongly suggest you use the TechBot format, however, to achieve the fastest turnaround.

When you e-mail us, the TechBot e-mail robot automatically opens a new Call Tracking Number and directly updates our technical support issue tracking database with the information contained in your e-mail. Almost as soon as it is received, your issue is ready for an engineer to investigate it. Additionally, using the TechBot format facilitates our ability to copy and edit details about your issue as it gets distributed internally within the company. For example, a support engineer can forward part of your message to a development engineer.

While we do offer telephone support, we cannot guarantee that an engineer will be available immediately to answer a question, so it may be necessary to call you back. We strive to respond as quickly as possible, but there is a great variation in demand for support services so the response time varies. A reply to any query should be provided within 24 hours (non-business days excluded).

We don’t recommend faxing source code, log files, etc., because they can be difficult to read and distribute among the tech support engineers.

What We Need From You

This section lists the information we need from you concerning your technical support issue. If you are calling us, make sure you have this information readily available. If you are e-mailing us, you can include it in a ZIP file.

1. Provide your customer number.

   If you don’t have your customer number available, include the name of your distributor or source from whom you purchased your software.

2. Include our call tracking number if this is a follow-up request.

3. Tell us which build of the product you are using by recording the creation date written to the log file.
4. Note which development environment you're using (Graphical VOS or CallSuite).
5. Specify the operating system and version you are using.
6. List the telephony hardware and driver configuration (and their version numbers) installed on your system.
7. Describe the problem including the steps to take in order to reproduce it. We also encourage you to provide the simplest test project that shows the problem (please do not include your full application unless it is very simple).
8. Include the log file produced when you run the application:
   - For Graphical VOS, include the VOS1.LOG file produced when you run the application (or include the VOS2.LOG if the problem shows up in that file). Use detailed logging when you generate the file and make sure to include at least the built-ins and drivers in the detail.
   - For CallSuite, include the Vbox.log file (or Vbox2.log file if the problem shows up in that file). Use detailed logging when you generate the log file, either by using the SetLogLevel methods with the LogLevel parameter set to LOG_Detailed or by setting LogForce=Yes in the VBocx.ini file.

   For more information about important log file entries and how to control the log output, see the chapter titled, "Troubleshooting" in this guide.
9. If your issue is especially urgent, make this clear in your report, and we will try to respond more quickly.

Contacting Us

- Send an e-mail to tech@Parity.com (this is the most efficient way to reach Technical Support).
- Send a fax to +1 (415) 332-5657.
- Call via telephone at +1 (415) 332-5656, then choose option 4.

Glossary

A&B Bits
A&B bits are used in digital environments to convey signaling information. A bit equal to “one” generally corresponds to loop current flowing in an analog environment, a bit equal to “zero” corresponds to no loop current—that is, to no connection. Other signals are made by changing bit values. For example, a flash-hook is sent by briefly setting the A bit to zero.

ABCD Bits
Signaling bits used on E-1 digital trunks to convey information about the state of a call. For example, ABCD bit values may be used to signal an incoming call, disconnect, seize and so on. They are closely analogous to the A and B bits commonly used on T-1 digital trunks. E-1 is a digital trunk standard used in many countries outside of North America. There are 32 channels (time-slots) in contrast to 24 channels on T-1, hence the faster bit rate of 2.048 MHz versus 1.544 MHz for T-1. Channels 0 and 16 are used to carry the ABCD bits and synchronization (framing) bits, hence only 30 channels are available for audio conversations. Thus, ABCD bits are carried out-of-band in contrast to the in-band robbed-bit scheme used by T-1.

ACD (see Automatic Call Distributor)

ActiveX
ActiveX controls are among the many types of components that use COM technologies to provide interoperability with other types of COM components and services. An ActiveX control is an example of an ActiveX object. ActiveX objects are units of executable code (usually stored in an EXE, DLL or OCX file) and data. Usually code is shared by all objects of a given type (class), but a separate block of data is created for each object. ActiveX objects are provided by ActiveX servers.

An application that uses an object is called an ActiveX client. Visual Basic is a well-known example of an ActiveX client. It is easy to load and use ActiveX objects such as ActiveX controls (also called custom controls) from within Visual Basic. VOS is also an ActiveX client. Using ActiveX objects in VOS is quite similar to using ActiveX objects in Visual Basic.

Adaptive Differential Pulse Code Modulation (ADPCM)
ADPCM is a speech coding method that calculates the difference between two consecutive speech samples in standard PCM-coded telecom voice signals. This calculation is encoded using an adaptive filter and is therefore transmitted at a lower rate than the standard 64 Khbps technique. Typically, ADPCM allows an analog voice conversation to be carried within a 32k-kbit/s digital channel; 3 or 4 bits are used to describe each sample, which represents the difference between two adjacent samples. Sampling is generally done 8,000 time a second (8 KHz). The Dialogic default ADPCM format, however, uses 4-bit sampling at 6 KHz.

ADPCM (see Adaptive Differential Pulse Code Modulation)

A-law Encoding (also see Mu-law Encoding)
A-law encoding refers to the assignment of a loudness value to an audio signal using an A-law translation table. Digital telephony equipment usually stores and transmits audio signal in 8-bit samples using PCM (Pulse Code Modulation) encoding. The value of the 8-bit sample corresponds to the amplitude (volume or loudness) of the audio signal at the time the sample was taken. A translation table is used to determine the amplitude given the sample value. The two tables in common use are the A-law table, which is widely used in Europe and Asia on E-1 trunks, and the Mu-law (m-law) table, which is used in the US, Canada, and several other countries.

Amplitude
The amplitude of a waveform or a signal is the distance between the high and low point of the wave. This determines the loudness or volume of the waveform or signal. Also known as wave height.

Analog (also see Digital)
Analog is derived from the word “analogous,” which means “similar to.” In telephone transmission, the signal being transmitted—voice, video, or image— is “analogous” to the original signal. In other words, if you speak into a microphone and see your voice on an oscilloscope, the two signals would look essentially the same. The only difference is that the electrically transmitted signal (the one over the phone line) is at a higher frequency. In correct English usage, “analog” is meaningless as a word by itself. But in telecommunications, analog means telephone transmission and/or switching that is not digital. Outside the telecom industry, analog is often called linear and covers the physical world of time, temperature, pressure, and sound, which are represented by time-variant electrical characteristics, such as frequency and voltage.

ANI (Automatic Number Identification)
Also known as Caller ID, ANI is a telephony feature that allows the receiver of a phone call to see who’s calling before picking up the incoming call. The digits of the caller may arrive in analog or digital form. They may arrive as touchtone digits inside the phone call or in a digital form on the same circuit or on a separate circuit. The person receiving the call needs some equipment in order to decipher the digits and to do something with them, such as add them to a database and display the matching client record on a screen so the receiver has access to the client’s information in order to provide faster customer service.

API (see Application Programming Interface)

Applications Programming Interface
A set of instructions provided by an operating system or device driver. Usually the functions are provided as subroutine calls that can be used by programs written in C or another programming language. An API gives users an interface to interact with, and from this interface the user can initiate contact with the underlying network services, telephone equipment (and in the case of CT ADE—Topaz) to request that low-level services and tasks be performed.

Application Generator (see CallSuite Application Wizard)

ARU (see Audio Response Unit)

ASR
Automatic Speech Recognition

Asynchronous Mode
A mode for calling driver functions where a function call to perform a time-consuming action like playing a file returns immediately to the application once the operation is started. Later, a message or event is sent that notifies the application when the operation is complete.

Audio
Audio refers to the sound you hear over the telephone which has been converted to electrical signal for transmission. Normal hearing range is between 15 and 20,000 hertz.

Audio Frequencies
Audio frequencies are those that the human ear can detect (usually between 15 and 20,000 hertz). Only those ranging from 30 to 3,000 hertz are transmitted through the phone, which is why the phone doesn't sound "Hi-Fi."

Audio Menu (also see Menu and Prompts)
An audio menu is the set of options spoken by a voice processing system. The user chooses a menu option by pressing a touchtone on the phone or speaking a word or two. Computer or voice processing software can be categorized as menu-driven or non-menu driven programs. Menu-driven programs are easier to use, but they can only present as many options as can reasonably be spoken in a few seconds. Audio menus are typically played to callers in automated attendant/ voice messaging, voice response, and transaction processing applications.

Audio Messaging Interchange Specification (AMIS)
AMIS is series of standards aimed at addressing the problem of how voice messaging systems produced by different vendors can network or inter-network. Before AMIS, systems from different vendors could not exchange voice messages. AMIS deals only with the interaction between two systems for the purpose of exchanging voice messages. It does not describe the user interface to a voice messaging system, specify how to implement AMIS in a particular system, or limit the features a vendor may implement.

Audio Response Unit (ARU)
An ARU device translates computer output into spoken voice. That is, it provides synthesized voice responses to dual-tone, multi-frequency signaling input. These devices process calls based on the caller's input, information received from a host database, and information carried with the incoming call (for example, the time of day). ARUs are used to increase the number of information calls handled and to provide consistent quality in information retrieval. For example, let's say you dial a computer and you hear, "If you want the weather in Chicago, push 123." You push the 1, 2, and 3 digits and then you hear the weather report for Chicago. What you heard was "spoken" by an ARU.

Audiotex (also Audiotext)
Audiotex is a generic term for interactive voice response equipment and services. Audiotex is to voice what online data processing is to data terminals. The idea is that you call a phone number and a machine answers, presenting you with several options such as, "For information on plays, push 1; for information on movies, push 2;" and so on.

Audiotst
Audiotst is an application that determines the number of voice boards present on your system. Then it plays a set of Wave files in various formats to each board to find out if those formats are supported. If they aren't, Audiotst reports an error message. You can use Audiotst with both Graphical VOS and CallSuite.

Automated Attendant
An automated attendant is a device that is connected to a PBX. When a call comes in, this device answers it and says something like, "Thanks for calling ABC Company. If you know the extension you'd like, press that number now and you'll be transferred. If you don't know it, press 0 and the live operator will assist you." Sometimes the automated attendant might give you other options, such as, "dial 3 for a directory." Some people react well to automated attendants, but others do not so it's always a good idea to let people know that the operator is automated.

Automatic Call Distributor (ACD)
ACD is a specialized phone system used for handling many incoming calls. Once used only by airlines, rent-a-car companies, hotels, etc., it's now used by any company that has a large number of incoming calls. An ACD performs four functions: recognizing and answering an incoming call, referencing its database to find out how to handle the call, routing the call to the appropriate place, and finally, routing the call to an operator.

Automatic Call Sequencer (ACS)
ACS is a device that handles incoming calls. Typically it performs three functions. First, it answers an incoming call, plays a message to the caller and then puts the caller on hold. Second, it signals the person taking the call which line to pick up. Finally, it provides management information such as how many abandoned calls there were, how long the longest person was kept on hold, how long the average "on hold" was, etc.

Automatic Call Unit (ACU)
An ACU is a device that places a telephone call on behalf of a computer.

Automatic Callback
Automatic Callback is a telecom term that describes a telephony feature in which a caller requests a "callback." For example, if a caller dials another internal extension and finds it busy, he can dial specific digits on his phone or press a special "automatic callback" button. When the person he tried to call ends her call, the phone system rings her number and the number of the original caller and then the phone system automatically connects the two lines. This feature saves a lot of time by automatically retrying the call until the extension is free.

B-Channel (also see D-Channel)
A B-Channel is an ISDN channel use to convey audio or data rather than signaling information. B stands for "bearer", as in the bearer of glad tidings.

Baby Bell
Baby Bell is a telecom term used to describe an RBOC (Regional Bell Operating Companies). RBOCs are responsible for local calls in the US following the break-up of AT&T.

Board Device
In the Dialogic API, some function calls reference board devices to set parameters or to perform operations that affect all channels on that board. A board device may be a "virtual board," that is, a single physical board may include more than one virtual board and hence more than one board device.

Cadence
In voice processing, cadence describes the pattern of tones and silence intervals generated by a given audio signal. Examples are busy and ringing tones. A typical cadence pattern is the US ringing tone, which is one second of tone followed by three seconds of silence.

Call Center
A call center is a place where calls are made and answered. Call centers usually have lots of people (agents), an automatic call distributor, a computer for order-entry, and lookup on customer orders. A call center may also have a predictive dialer for making lots of calls quickly (see Predictive Dialing).

Call Completion
Call completion indicates that a call has "gone through." When a call has been completed, there is an unbroken (complete) circuit made between the caller and the recipient of the call. This circuit is known as the talk path.

Call Progress Analysis (CPA)
CPA is the automated determination by a piece of telecommunications equipment as to the end result of dialing a number. For example, the result of the analysis might be a busy tone, ringing at the other end but no answer after a pre-set number of rings, or an answered call. The analysis also involves detecting various call progress tones which are generated by the telephone network as the call is put through.

Call Progress Monitoring (also see Call Progress Analysis)
Closely related to call progress analysis, call progress monitoring must be active during the entire length of a conversation. For example, when a call is placed across a PBX or to a country that does not provide for loop current drop disconnect supervision, the equipment listens for a "re-order" or dial tone to determine that the caller hung up. This is classified as call progress monitoring because it must take place during the entire call, not just as a number is being dialed or while a transfer is taking place.

Call Progress Tone
Call progress tones are sent from the telephone switch to inform the caller of the status of the call. Examples are the dial tone, busy tone, ringback tone, and error tone.

Call Supervision
Call supervision is a general term describing either call progress analysis or call progress monitoring (see Call Progress Analysis and Call Progress Monitoring).

Caller ID (also see ANI)
Caller ID is a service that displays the calling party's telephone number on a special display device. Caller ID is also referred to as ANI.

Calling Tone
A calling tone is the “whistle” tone (1,100 Hz) that a fax machine makes to inform the caller that it is ready to receive a transmission.

CallSuite
The CallSuite implementation of the CT Application Development Environment is a collection of ActiveX components that let you add telephony features to your existing systems or create applications from scratch. CallSuite custom components can be used in any Windows development environment that supports ActiveX components.

CallSuite Application Wizard
With Microsoft Visual Basic and Visual C++, CallSuite Wizard can generate a complete application to get you jump started.
In Visual Basic, this feature is automatically provided when you select CallSuite Wizard from the Add-Ins menu in a project where CallSuite Wizard has not previously been used. The AppWizard will lead you step by step through the process of creating a new project. When the project has been created, it will be ready to run and either accept incoming calls or dial outgoing calls.

In Visual C++, you will find CallSuite AppWizard in the standard AppWizard list for creating new projects.

CallSuite Wizard
CallSuite Wizard writes and edits computer telephony program source code for you. You can think of CallSuite Wizard as a highly specialized code editor. The code that CallSuite Wizard creates uses Parity Software CallSuite ActiveX controls to perform computer telephony actions such as touch-tone menus.

You don't have to use CallSuite Wizard to write CallSuite code. If you wish, you can use Basic, C++, or another language supporting ActiveXs to write your own code. However, CallSuite Wizard can make it easier for you: you don't have to remember properties or methods, and you can't make a syntax error.

Central Office (CO)
In North America, a CO is a location that houses a switch to serve local telephone subscribers. Sometime the words central office are confused with the switch itself. Outside the US, a CO is more commonly known as a public exchange.

Channel
A path of communication, either electrical or electromagnetic, between two or more points. Also called a circuit, facility, line, link, or path.

Channel Device
In the Dialogic API, most function calls address channel devices. A channel device typically processes the audio from a single telephone connection.

Co-Articulation
When speaking a pair of words such as “seven nine” or test tube”, people generally omit the consonant that starts the second word, saying something like, “seven’ine” or “test’ube.” This phenomenon is called co-articulation.

Conference
A conference is a configuration where three or more callers are able to talk to each other. Some parties may be in a “listen-only” mode where they can hear other parties but cannot speak to them.

Continuous Speech
Continuous speech is a telecom term used to describe speech that is made up of a series of utterances made in relatively quick succession with co-articulation. Continuous speech is usually applied to the capability of a voice recognizer to recognize words from this type of speech.

CT ADE
The Computer Telephony (CT) Application Development Environment (ADE) is a set of development tools and programming interfaces that help shorten both your time to market and your time to revenue by making it quick and easy to build robust, portable CT applications. The underlying Topaz architecture eliminates the need for developers to take time learning current and new hardware and API protocols. The Topaz architecture is an abstraction layer that sits on top of an API and performs low-level CT tasks, letting you focus on building your applications without worrying about these low-level operations. You can access Topaz using either of the two included programming environments: CallSuite or Graphical VOS.

D-Channel (also see B-Channel)
D-Channel is the data channel on an ISDN trunk. On a Primary Rate Interface ISDN trunk, there are 24 channels (time-slots) just as on a usual T-1 trunk. One or more channels (called D-channels) are used to convey information such as dialed digits, ANI and DNIS information, routing and billing codes, depending on the type of ISDN service used. Therefore there are only 23 or fewer audio channels (called B-channels) available. This is a form of out-of-band signaling protocol that can give faster responses and more flexible services than the in-band DTMF and robbed-bit technology in wide use today.

Debugger
The Debugger is a source-code level graphical debugger that lets you test and troubleshoot VOS programs through the development phase and after installation and deployment. You can use this tool to debug many programs in a single session, check program variables, and check the current state of your application without interrupting the calls in progress. Debugger is for use with Graphical VOS only.

Digital (also see Analog)
In telecommunications, recording, or computing, digital refers to the use of binary code to represent information (see Pulse Code Modulation). Alternatively, analog signals like voice or music are encoded digitally by sampling the voice or music analog signal many times a second and assigning a number to each sample.

Recording or transmitting information digitally has two major benefits. First, the signal can be reproduced precisely. This is important because in a long telecommunications transmission circuit the signal progressively loses its strength and begins picking up distortions, static, and other electrical interference “noises.” In analog transmission, the signal, along with all the garbage it picked up, is simply amplified. In digital transmission however, the signal is regenerated (that is—squared off) to what it was identically. It’s put through a little “Yes-No” question. Is this signal a “one” or a “zero?” Then the new signal is amplified and sent along its

way. This makes digital transmission much “cleaner” than analog transmission. The second major benefit of digital is that the electronic circuitry that handles digital is becoming less expansive and more powerful.

Digital Trunk
A digital trunk is a generic term for a telephone connection that uses digital rather than analog transmission technology. Common examples are T-1 in the US and E-1 in Europe.

Digitization
Digitization is the process of converting an analog signal to a digital signal by measuring the value of the analog signal at regular intervals and converting this to a numerical value (called a sample).

DLL (see Dynamic Link Library)

DNIS (Dialed Number Identification Service)
DNIS is a feature of 800 and 900 lines that tells the receiver of the call which number the caller dialed. For example, let’s say that you subscribe to several 800 numbers. You use one line for testing your advertisements on TV stations in Phoenix, another line for testing your advertisements on TV stations in Chicago, and a third line for Milwaukee. Next you get an automatic call distributor and you terminate all the lines in one group to your ACD because it’s cheaper and more efficient to manage this single group of incoming lines. You ask all of your people to answer all of the calls, regardless of where they originate. Now you need to know where each of the calls are coming from, so you ask your long distance carrier send you the DNIS for each of the calls—this is the number each person dialed to reach you. Depending on the technical arrangement you have with your long distance carrier, the DNIS digits may come to you in-band or out-of-band, ISDN or data channel, etc.

Dongle (see Hardware key)

DTMF (Dual Tone Multi-Frequency)
DTMF is another term for touchtone dialing. When you dial a digit, a tone is produced. This tone, also called a signaling digit, is actually a combination of one high-frequency tone and one low-frequency tone. Two tones are used together to signify a digit instead of just one in order to prevent the possibility of a human voice being mistaken for a tone.

DTMF Cut-Through
DTMF cut-through is the ability of CT equipment to respond immediately to a received touchtone even when a voice prompt is being played.

Dynamic Link Library (DLL) (also see Runtime Link Library)
A DLL is a library that gets linked to application programs when they are loaded or run. The same block of library code can be shared between several tasks rather than each task containing copies of the routines it uses. The executable is compiled with a library of “stubs” which allow link errors to be detected at compile-time. Then, at run time, either the system loader or the task’s entry code must arrange for library calls to be patched with the addresses of the real shared library routines, possibly via a jump table.

Editor (also see FlowCharter)
The Graphical VOS Editor is a syntax-highlighting editor designed specifically for VOS language source code. As you edit code, the Editor highlights and colorizes different VOS elements, making it easy to follow the flow of your code. You can use the Editor to create new VOS source and function files, or to edit existing ones.

Event
Events are actions that you can write code to respond to. For example, there are several events that can be triggered by CallSuite components, including the VoiceBocx IncomingCall and CallerHangup events. Many events have corresponding methods or functions that wait for the specified action to take place.

FlowCharter (also see Editor)
VOS FlowCharter is a graphical tool that lets you create and edit telephony applications. Instead of writing source code, you draw telephony applications by arranging cells (modules of code that perform specific CT functions) into a flowchart layout and linking them in the order you want the application to flow.

Function
A function is a sequence of statements that has a name and produces a value. A function can have one or more “arguments,” also called “parameters.” Arguments are values that are copied into the function. Inside the function, arguments are referred to by names that behave much like variables except that they may not be assigned values.

Graphical VOS
You can create and deploy telephony systems with Graphical VOS, a complete development environment that includes Topaz, the VOS Language Compiler (VLC) and runtime engine, FlowCharter, Editor, Runtime Link Libraries, Debugger, and SimPhone. Graphical VOS provides an environment in which you can write, test, debug, and run your CT applications in a Windows environment. The VOS engine is responsible for executing the applications built with the VOS Language Compiler.

Grammars
A grammar is a set of rules that accompany a language and determine which prompts to play during application runtime. The grammar section of the language profile is labeled with the following start and end tags:
<grammar>
</grammar>

Topaz begins processing a phrase element at a specified label in the grammar, comparing the phrase element's Value to a series of patterns. When it finds a matching pattern, Topaz uses the rules that correspond to that pattern to determine which IPF prompts to play.

Each grammar entry follows this format:
[Label], Pattern, Rule

Hardware key
A hardware key, sometimes called a dongle, must be present for VOS to run with full functionality. This key allows VOS, VLC, and Graphical VOS to run on up to two ports of any Topaz Resource type.

Indexed Prompt File (IPF)
The Indexed Prompt File (IPF) contains VOX files, also known as prompts, which are recorded words or phrases. Compiled with an .ipf extension, each IPF begins with a header that includes an indexed listing all of the included prompts. The IPF is a profile include file that lets you specify which Indexed Prompt Files are opened when Topaz starts (also see Profile Include Files). These IPFs can be played directly, and they can also be used by the Topaz International Phrases.

IPF (see Indexed Prompt File)

Interactive Voice Response (IVR)
A telecommunications system, prevalent with PBX and voice mail systems, that uses a prerecorded database of voice messages to present options to a user, typically over telephone lines. User input is retrieved via DTMF tone key presses (see DTMF). When used in conjunction with voice mail, for example, these systems typically allow users to store, retrieve, and route messages, as well as interact with an underlying database server which may allow for automated transactions and data processing.

IVRs serve as a bridge between people and computer databases, interactive voice response systems (IVRs) to connect telephone users with the information they need from anywhere and at any time. These systems have been around for more than a decade, and today they are used to support stock trade transactions, make travel arrangements, and manage bank accounts.

Most of today's IVR and transaction-processing applications employ a touch-tone or dual-tone multi-frequency (DTMF) user interface. However, applications that allow callers to use their own voice rather than DTMF inputs to complete transactions are rapidly emerging as the latest innovation in telephony-based remote self-service.

Method
Methods are used like functions to implement actions. For example, you can use the Send method of Microsoft MAPIMessages control to send an e-mail message. In CallSuite, most of the actions that you want to perform can be enabled using methods. For example, the SetLogLevel and GetLogLevel methods set and check the level of detail to use when logging fax operations.

Mu-law Encoding (also see A-law Encoding)
Mu-law encoding refers to the assignment of a loudness value to an audio signal using a mu-law translation table. Digital telephony equipment usually stores and transmits audio signal in 8-bit samples using PCM (Pulse Code Modulation) encoding. The value of the 8-bit sample corresponds to the amplitude (volume or loudness) of the audio signal at the time the sample was taken. A translation table is used to determine the amplitude given the sample value. The two tables in common use are the A-law table, which is widely used in Europe and Asia on E-1 trunks, and the Mu-law (m-law) table, which is used in the US, Canada, and several other countries.

multithreaded application
In CallSuite, a “thread” is an execution path through an executable program. Most programs have only a single thread. 32-bit Windows allows programs to start additional threads. These “background” threads (as opposed to the “primary” or main thread, which usually controls the user interface) can be used for tasks such as printing or recalculating. Running on a separate thread allows you to continue working in the program. Without a background thread, you may find yourself waiting and looking at an hourglass cursor for the duration of the operation.

NetHub Plus (communication technology)
Available both as an ActiveX control and as a VOS RLL, NetHub Plus is an enterprise component that supports communication (messaging) between threads in a single process, between processes in a single PC, and/or between nodes on a local or wide-area network.

Operator Intercept
When an invalid number is dialed or an error condition occurs on the network, an operator intercept may occur. In the US, SIT tones are heard and then followed by a recorded message explaining the problem.

Out-Of-Band Signaling
Signaling that is separated from the channel carrying the information—the voice, data, video, etc. Typically the separation is accomplished by a filter. The signaling includes dialing and other supervisory signals.

PABX
Private Automatic Branch Exchange. Originally, PBX was the word for a switch inside a private business (as against one serving the public). Such a PBX was typically a manual device, requiring operator assistance to complete a call. Then the PBX went modern (that is, automatic) and no operator was needed any longer to complete outgoing calls. Instead, you can simply dial 9. Thus it became a PABX, Now all PABXs are modern and a PABX is now commonly referred to as a PBX.

PBX
Private Branch Exchange A private (meaning that you--not the phone company, own it) Branch (meaning it is a small pone company central office), exchange (a central office was originally called a public exchange, or simply an exchange). In other words, a PBX is a small version of the phone company’s larger central switching office.

Personality
In text-to-speech, a “personality” describes a single voice that can be used to speak text. Personalities on your system can use different speech engines, speak different languages, or simply speak with different accents.

Phraser.exe
Phraser.exe is a utility that lets you test specific words and phrases from your language. This is especially useful when you are creating a new language and you want to test cases where the outcome may be hard to predict, such as the phrase ‘twelve midnight’.

POTS (Plain Old Telephone Service)
POTS is the basic service supplying standard single line telephones, telephone lines, and access to the public switched network.

Predictive Dialing
An automated method of making many outbound calls without people and then passing answered calls to a person as the calls are answered.

Profile ID (see Topaz Profile ID)
(Also called a TZP file) Entries in the Profile are called Profile IDs and each has a name and a value. The name of an entry is similar to a path name in a file system. All names begin at the root, which is designated by a back-slash character (\).

Profile include file
Also called TZP files, profile include files are actually text files containing information that gets added to the Topaz Profile the next time Topaz is loaded. For example, you can use a profile include file to add a new language to Topaz. You can create new profile include files or edit existing ones using any text file editor such as Notepad or the Graphical VOS Editor.

Property
A property is a data value similar to a variable and is generally used for attributes of an object that tend to remain fixed. Typical object properties might be BackColor and Font for an object that is displaying a button on the screen. Properties are specified using the name of the object and the name of the property separated by a period.

Resource
Topaz is built around the concept of a Resource. A Topaz Resource produces and/or consumes a single stream of audio. There are six groupings of Topaz Resources: Trunk interface Resources, Media Resources (player / recorders), Fax Resources (sender / receivers), Text-to-speech Resources, Voice recognition Resources, and Conferencing Resources.

Resource index number
When VOS and Topaz are started, each Resource is assigned a Resource index number, from 0 to the total number of that type of Resource on your system minus one.

RegID
RegIDs are values used internally by Topaz. Some RegIDs describe system details and are stored in the Topaz Profile. These RegIDs are called Profile IDs (see Profile ID). The contents of the Profile can be viewed with the TopazProfile.exe -L command, which generates a text file that lets you view the contents of the Topaz Profile. Each entry in the generated file represents a Profile ID.

Topaz uses other RegIDs internally to access technology-level information. These RegIDs do not appear in the Topaz Profile, but can be used with the Get/Set functions to retrieve information about the system or to issue a Technology-specific command at runtime.

RLL (see Runtime Link Library)

Routine
A Routine is a series of source code statements that can be invoked by a single name. In Visual Basic, a routine is actually a subroutine. In C++, a routine is actually a function. Other languages also have different names for the same concept. In CallSuite however, when Routine is spelled with a capital R, this denotes a CallSuite Wizard Routine.

CallSuite Wizard defines several types of Routines that cover the operations most commonly required in call processing applications.

Resource scanner (see Topaz scanner)

Runtime Link Library (RLL) (also see Dynamic Link Library)
Runtime Link Library (RLL) add-ins are extensions that expand VOS functionality. Graphical VOS comes with a number of RLLs implemented as Dynamic Link Libraries (DLLs) that let you load executable code modules on demand to be linked at runtime. RLL functions can be called from a VOS application like other VOS functions.
For example, the Sockets RLL lets you send and receive messages to VOS tasks running on remote PCs that are connected via an IP network.

SAPI
Microsoft Speech API (SAPI) is a platform for adding speech recognition and text-to-speech capabilities to computer telephony applications.

Seize
Seize is a computer telephony term that refers to the process of getting access to a phone line prior to making a call. On a home phone, the action of taking the phone off-hook by lifting the handset is known as a seize.

SimPhone
SimPhone works together with the Topaz engine, the VOS runtime engine, and your PC's speaker and microphone to simulate a telephony board and phone line so that you can develop and test VOS applications without requiring a telephony board. This can be helpful in situations where you don't have access to a telephony board. For example, if you are on a business trip you may only have a laptop computer available as your development environment.

Speech-To-Text (also see Voice Recognition)
Speech-To-Text is another name for Voice Recognition.

Technology (Topaz Technology)
In Topaz, a Technology is a specific hardware and API combination that creates a given Resource type.
Topaz was designed for API transparency so the same set of functions and methods work under all supported telephony APIs and all supported trunk types—but some features are available only under certain Technologies. For example, GlobalCall systems transmit billing rate information, but other trunk interface APIs (LSI, MSI, etc.) don't use protocol-defined billing rates. Consequently, a VoiceBocx "GetTrunkBilling" method, which couldn't be ported from the R4GcTrunk Technology to other Technologies, would not be API transparent.

Technology Type
Technology Type/Index pairs (TTIPs) are used in the Topaz Profile to set up both listen restrictions and listen additions for Free Listen Resources. A TTIP specifies the Technology type and Technology index number of the Resource that the free listen Resource can listen to.

Text-To-Speech (TTS)

TTS is the creation of audible speech from computer readable text. Text-To-Speech (TTS) technology enables a VRU to convert textual data into speech output so that a voice-processing application can convey information to the caller that was not pre-recorded. Also called speech synthesis, TTS is particularly useful when you need to be able to communicate a wide array of unpredictable information to callers.

For example, the number 123 can be spoken as the four elements: “One” “Hundred” “Twenty” “Three”. With a pre-recorded vocabulary of less than 40 elements it is possible to speak any number less than a billion billions. With a few more pre-recorded elements, more elaborate messages such as: “You have” “Eighteen” “New messages” can be strung together. However, Text To Speech more usually refers to the ability to speak almost any text, such as a newspaper article.

TTS also facilitates efficient information storage. One hour of speech consumes ten megabytes or more of storage in the form of audio files. The same information would typically require 50Kb or less in the form of ASCII text, just 0.5% of the space required for the audio files.

Touch Tone (also see Dual Tone Multi-Frequency)

Touch tone is a former trademark once owned by AT&T to describe a tone used to dial numbers. Now touch tone is used to describe any phone that has “push-button” numbers.

Topaz

Topaz is a collection of software modules. The core component is a thin layer of code that mediates between your programming commands and the underlying Resource API. For example, if you invoke the VOS function TrunkAnswerCall, Topaz determines which trunk interface is being used, whether it is legal to make this call, and then chooses the appropriate API function to use.

Topaz.ini file

The Topaz.ini file is used to specify various configuration parameters (such as which Telephony technologies to exclude from the Topaz Profile scan) that Topaz reads each time it is invoked on your system. Before you change any entries in Topaz.ini, you must close all instances of VOS or CallSuite and any CT ADE utilities that use Topaz (such as TopazProfile.exe or Phraser.exe) so that the changes can take effect.

TopazProfile.exe (see Topaz scanner)

Topaz Profile ID

Entries in the Topaz Profile are called Profile IDs. Each Profile ID has a name and a value. The name of an entry is similar to a path name in a file system. The contents of the Profile can be viewed by using the TopazProfile.exe -L command, which generates a text file that lets you view the contents of the Topaz Profile. Each entry in the generated file represents a single Profile ID.

Topaz scanner

The Topaz scanner is a utility (called TopazProfile.exe) that runs on your system and extracts information about any installed telephony boards as well as their configuration details. The Scanner then builds a database called the Topaz Profile and places the extracted information into it. The Resource Scanner must be re-run each time you change your system's hardware or driver configuration.

TZP file (see Profile include file)

Vocabulary

In CT ADE, a vocabulary is a set of pre-recorded audio files used to synthesize phrases such as, "You have five new messages." The vocabulary for this message is probably made up of three fragments: You have / five / new messages.

Voice Board

(Also called a voice card or a speech card) A voice board is a computer add-in card that performs voice processing functions. All voice boards have several important characteristics such as a computer bus connection, a telephone line interface, and typically a voice bus connection. They also usually support going on and off-hook, notification of call termination, sending flash hook, and digit dialing. All voice boards support at least one operating system.

Voice Messaging

Voice Messaging refers to a number of actions that include the recording, storing, playing back, and distribution of phone messages.

VOS Language Compiler (VLC)

You can use the VOS programming language to write and maintain standalone computer telephony (CT) applications. Similar looking to C, VOS was specifically designed to support CT technology. VLC translates VOS source code into a VOS executable at runtime, which can then be used by the VOS runtime engine. The coding process is simpler than coding in C or C++ because many of the application details are handled by the VOS engine. This also makes maintenance and modifications easier to manage.

Voice Recognition (VR)

Voice Recognition is the ability of a machine to understand human speech. When VR is applied to telephony environments, the limited bandwidth (range of frequencies transmitted by a telephone connection) along with factors such as background noise and the poor quality of most telephone microphones limit the ability of current technology to recognize spoken words. Typical systems are able to recognize standard vocabularies of sixteen or so words, including yes, no, stop, and the names of digits.

Voice Response Unit (VRU)

(Also called Interactive Voice Response Unit or IVR) a Voice Response Unit can be thought of as a voice computer. While a computer has a keyboard for entering information, a VRU uses the touchtones from a remote telephone. While a computer uses a screen to display results, a VRU uses a digitized synthesized voice to "read" the screen to the distant caller. An IVR can do anything that a computer can, from looking up train timetables to moving calls around an

automatic call distributor (ACD). The only limitation of an IVR is that you can't present as many alternatives on a phone at one time as you can on a screen because callers can remember only small chunks of information.

WaveTest

WaveTest is a command-line utility that you can use to scan your system for installed wave device. It also enables you to find out what each wave device does. You can use WaveTest with both Graphical VOS and CallSuite.

Word Spotting

The term Word Spotting refers to the ability of an automated voice recognition device to pick out certain selected words from "background" conversation. For example, the recognizer might pick out "no" from "No thanks,"or "five" from "Give me five please."

Index