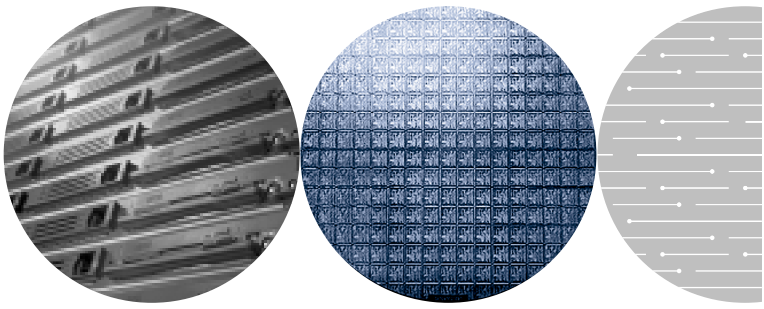
Application Note

Intel® NetMerge™ CT Application Development Environment Concepts and Facilities

Intel in Communications



Contents

Figures

Tables

Introduction

This paper will help you understand the value of the Intel® NetMerge™ CT Application Development Environment (CT ADE) to your development projects. It should be of benefit if you:

- Normally write telephony applications in C/C++ using the hardware APIs
- Have used earlier Intel products known as VOS and CallSuite
- Have used other application development tools
- Are new to telephony applications

Computer Telephony

To begin, it might be useful to remind ourselves what we are trying to do with this technology, in whatever form it comes.

Our applications need to:

- **Detect and answer** incoming calls
- **Place outbound calls and detect results** (busy, not answered, answered by voice, answered by machine, etc.)
- **Converse with callers** (audio going to caller, telephone tones or voice coming from caller)
- **Exchange data with callers** (to and from a database)
- **Control calls** (conference, disconnect, transfer, hold, retrieve)

Next, these basic operations need to be performed:

- **Within networks of many different protocols** (loop-start, T-1/E-1 CAS, ISDN, IP) with variations by country
- **Within installations of many sizes** (2 analog ports, 500 ports in one chassis, multiple cooperating chassis)
- **Under operating system control** (Windows*, Linux*, Unix*)
- **With links to system management monitors and managers**

Finally, all of these tasks must work in harmony with business logic that analyzes inputs and directs actions and outputs.

The options for creating such applications are:

- **Packages** — If your intended application performs fairly common actions, you might find complete packages that do it 90% the way you want it and can be configured or customized to meet the final 10%.Voice mail and interactive voice response (IVR) systems are popular applications in this category.
- **Frameworks** — Less specific solutions are available that provide a telephony container for your business logic. Call control, monitoring, and management components provide a prefabricated structure into which your application work is added. These range from script languages to drag-and-drop function blocks.
- **Object libraries** — Telephony functions can be delivered as class objects to be used in non-telephony languages such as C++*, Delphi*, and Java*. The developer then provides the business logic and the application design using one of the standard languages from which the library functions can be called. Objects and frameworks are sometime referred to as middleware.
- **Direct APIs** — This is the interface provided by the device manufacturer to control the hardware. It comes in flavors specific to the operating system in which it lives. Here the application programmer deals not only with the business logic and program design, but also with the handles, events, states, and masks of the telephony hardware devices.
- **Speech APIs** — Besides interfacing to hardware, there are direct APIs for other technologies such as speech synthesis and speech recognition. Again, these can be controlled directly, usually in C/C++, or abstracted to service functions in high-level languages, libraries, and packages.

Which is best? The answer depends on not only the application and the products available, but also on your own company or department experience and mission. Issues that distinguish these alternative approaches include:

- **Packages** – If it is built with all the features you need, a turnkey solution can be rapidly deployed. If not all your requirements are implemented, it can be difficult to customize or extend ready-made applications, depending on how well they are designed for change. Performance and scale can be problems with general solutions, so benchmarks before purchase are a must.

- **Frameworks** –These come in different forms that are more or less flexible when not all of the framework's assumptions align with your application objectives. While these can make the application programming easier for the most common telephony operations, some products do not provide access to less common features of the hardware. Like packages, performance can be a significant issue depending on how general the solution or optimized the design.
- **Object Libraries** – Because they make fewer assumptions about what the application will do, these components are more flexible than packages and frameworks. By abstracting the specific interface calls to common forms, they ease portability among like technologies (e.g. SpeechWorks* and Nuance* for automatic speech recognition [ASR], ISDN and CAS network protocols, or DM/IP boards and Host Media Processing for IP connectivity).
- **Direct APIs** – These are certainly the most comprehensive tools available for controlling the hardware or speech engine. The tradeoff is the demanding level of detail that must be accurately managed. The costs of training and implementation for this choice need to be considered, not only for the initial version of the application, but also for the inevitable changes in the underlying technologies.

Intel® NetMerge™ CT Application Development Environment

Intel provides three of the alternatives listed above:

- A direct C/C++ API (known as R4)
- A telephony framework
- An object library

The latter two are packaged as Intel® NetMerge™ CT Application Development Environment (CT ADE). (You can find more information on R4 at http://www.intel.com/network/csp/products/indx_aet.htm.)

The rest of this paper describes the concepts and facilities of the two CT ADE programming platforms:

- **Application Development Language (ADL)** — A procedural language, with optional graphical interface, incorporating many telephony application building blocks.
- **Application Development ActiveX Objects (ADX)** — A library of methods with a COM interface that can be incorporated into Windows visual development languages like C++, Visual Basic*, Delphi, as well as .NET languages C# and VB.NET.

At the heart of these two platforms is the Resource Manager, the foundation code that performs all the technology-specific interactions.

Resource Manager

The core component of the Resource Manager is a thin layer of code that mediates between your programming command (like Play in ADX or MediaPlay in ADL) and the underlying device API. For example, if you invoke the ADL function TrunkAnswerCall, the Resource Manager determines:

- Which trunk interface is being used
- Whether it is legal to make this call (Has an incoming call been signaled?)
- Which API function to use: *dx_sethook* (R4 analog), *cc_Answer* (R4 PRI), *gc_Answer_AnswerCall* (R4 Global Call), etc.

To achieve this device abstraction, we need to define the devices and all of their characteristics.

Resources

Resource Manager is built around the concept of a resource. Technically, a resource produces and/or consumes a single stream of audio. Resources fall into these categories:

- Trunk interface
- Media (player/recorder)
- Fax (sender/receiver)

- Text-to-speech (TTS)
- Voice recognition
- Conferencing
- Some examples should help clarify these ideas.

First, consider an Intel® Dialogic® D4PCI voice processing board with four channels. Each channel is viewed by Resource Manager as having two resources: a trunk resource and a media resource. The trunk resource corresponds to the phone line connector. The media resource corresponds to the VOX device that can play and record sound files.

Two audio streams are managed by the channel. The caller's voice (as well as touch-tone digits and other audio) follows this path:

Caller → Analog phone line → Trunk resource → Media device (VOX/Wave recorder)

In the other direction, the audio stream originates at the VOX/Wave player resource and is sent to the phone line through the trunk interface:

Media resource (VOX/Wave player) → Trunk resource → Analog phone line → Caller

As this example shows, the audio stream output from one resource can sometimes be used as input to another resource. In the simple case of a D4PCI board, the media and trunk devices are hard-wired so that the output from one is input to the other and vice versa. With higher-end devices, it may be possible to control this routing on the fly (e.g., with devices on an SC Bus or CT Bus).

For another example, consider an Intel® Dialogic® D41JCTLS converged communications board. This board has four analog trunk interfaces (called LSI devices) and four player/recorders (VOX devices). Resource Manager considers an LSI to be a trunk resource and a VOX to be a media resource. From the Resource Manager's point of view, this board is similar to the D4PCI board. The main difference is that the routing of the resources can be changed via the CT Bus.

A T-1 PRI ISDN interface board such as the Intel® Dialogic® DTI240SC board is treated by the Resource Manager as 23 trunk resources, one for each voice (B) channel. In other words, a trunk resource corresponds to a single T-1 timeslot. The signaling (data, D) channel on an ISDN circuit is not a Resource Manager resource at all. (It does not produce or consume any audio.) The existence and management of the D channel is, for most purposes, hidden from the Resource Manager user. For example, if the T-1 suffers an LOS condition (loss of signal, or D channel down), then each of the 23 Resource Manager trunk resources for the B channels transitions to a network down state (more on resource states later). If a B channel wishes to dial a call, which requires sending a data packet on the D channel, the Resource Manager command is exactly the same as for making a call on an analog trunk by dialing DTMF digits.

Resource states marshal low-level technology events and command results into operational resources states. For the most part, you can write your applications without following the underlying state changes. However, when needed, these logical call progressions can expose some unexpected conditions.

Resource States

For every kind of resource (trunk, media, fax, TTS, voice recognition, conference), Resource Manager predefines a number of states. For example, a trunk resource can be ringing or disconnecting and a media resource can be playing or recording.

The state of a resource can change in two ways: unsolicited (as a result of an external event) or as a result of a Resource Manager function call.

An example of an unsolicited state change is a trunk resource that is idle transitions to a ringing state when an incoming call is signaled.

An example of a state change due to a function call is a media that is idle transitioning to a playing state in response to a play command.

Functions that can change the state of a Resource Manager resource are known as commands. For example, the ADL function MediaPlay and the ADX method PlayWave invoke Resource Manager commands.

Resource Manager strictly enforces the following rules:

- Each command has a given set of states where it is legal to issue the command. Most often, the resource must be in the idle state for the command to be legal. Also, for most commands there is only one state in which the command can be issued. Important exceptions are the Abort and Reset commands.
- If the command fails, the state of the resource is left unchanged.
- If the command succeeds, the state of the resource is immediately changed to a new state. For each command, this state is fixed so that the application can be sure without having to check that if the command succeeds, the resource has transitioned to a given new state. For example, if Play does not return an error, the resource is guaranteed to be in the playing state.

Every resource starts out in the opening state. ADL and ADX normally hide this state from you, so you are unlikely to find a resource in this condition. When device initialization is complete, it transitions to the idle state and is then ready to accept a command.

These states and commands are common to all Resource Manager resources of all types:

- Opening state
- Idle state
- Reset command
- Resetting state
- Abort command
- Aborting state

Note that the Resource Manager definition of idle may differ from an “idle” condition defined by the underlying API. For example, in the R4 API, an LSI device is considered to be idle unless it is in the process of going on- or off-hook. It is considered to be “busy” if the attached VOX device is playing or recording. The R4 API does not have the ability to track the logical state of the call (connected or not) except through indirect means such as checking for the current hook switch state plus the presence of loop current on the line. This is not fully reliable since there may be short glitches in loop current without disconnecting a call. In Resource Manager, by contrast, the Idle state means that there is definitively no call in progress. A live call puts the trunk resource into the connected state.

Profile

The Resource Manager Profile is a database that is quite similar to the Windows Registry. The Profile stores the following information:

- Details of all installed hardware devices as determined by the Resource Scanner (described below)
- User-supplied hardware configuration information which cannot be determined by the Resource Scanner
- User-configurable options, such as the default language for speaking values (English, Spanish, etc.)

The Profile is treated as read-only by running applications (ADL and ADX) and should be fully initialized before these applications are started.

No dynamically changing information, such as the current state of a device, is stored in the Profile.

Entries in the Profile have 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 (\). For example, a Profile entry much used by Resource Manager code internally is:

```
\TechCount=4
```

The value of TechCount is the number of different Resource Manager technologies installed in this PC (a technology is a specific hardware and API combination (e.g., R4 VOX).

Internally, the Profile does not store the value names. All value names are actually stored as integers. The set of integers used for value names is predefined. Resource Manager utilities convert between the integer values used internally and the string names to display the Profile in a form convenient for human readers. This allows for more efficient lookup algorithms and, hence, faster access to the Profile when Resource Manager applications are running. The allowed integer values for value names are sometimes called RegIDs (Registry Identifiers) because the Resource Manager code internally refers to the Profile as the Registry.

Unlike the Windows registry, user application code has no direct access to the Profile and cannot create new entry names. The Profile is for internal use by Resource Manager only.

Resource Scanner

Determining the installed hardware and driver configuration is an important and often tricky task. Traditional APIs have unique and unequal functions to query the installed configuration. Often, important information is simply unavailable.

Building the hardware/driver configuration database in the Resource Manager Profile is a two-step process:

1. First run the Resource Scanner, which extracts all available configuration information from the available device APIs. The Resource Scanner must be rerun each time the hardware or driver configuration changes.

2. The second step "manually" updates the Profile. This step adds information that cannot be automatically determined and so must be provided by the user. For example, for an E-1/R2 trunk, the user must specify which country-dependent parameters should be used for the R2 protocol. For an analog trunk, the user must specify whether caller ID is available. For a T-1 CAS trunk, the user must specify which Global Call protocol is used to interface to the PBX or CO switch.

Building the device scanning process into the application at runtime presents a number of drawbacks:

- The scanning may be quite time-consuming, which slows launching your application. By running the Resource Scanner only when needed, application startup is faster.

- New driver releases sometime deprecate old APIs and/or introduce new APIs for configuration information. By separating scanner code from runtime code, we can more easily provide up-to-date scanners that adapt to the latest software releases.

Technology-Specific Access

The Resource Manager delivers high-level abstractions so that application programming can be API transparent (i.e., the same set of functions works under all supported telephony APIs and all supported trunk types). Some less common aspects are available only with certain technologies. For these cases, the lower-level API is exposed through instruction or data identifiers (RegIDs) and commands to execute the instruction or access the data (e.g., GetInt [get integer value] and SetInt [set integer value] and counterparts for boolean and string values).

These functions are not usually required to program applications using the telephony hardware, but are available for more advanced operations. The functions translate directly to the execution of the hardware API functions, as shown in the descriptions below.

Example: Set the retry strategy for fax operations:

SetInt   R4GrtFaxRetryStrategy
Use this REGID to set the m_gfqRecord.retry_strategy API element.

Example: Set the digit detection feature on a DCB conference device:

SetInt   R4DcbConfEnableDigitDetection
Use this REGID to directy access the dcb_setdigitmsk(handle, ConfId, Value, CBA_SETMSK) API function to enable and disable digit detection.

Example: Enable any device mask for a digital trunk using Global Call:

SetInt R4GcEnableMask

Use this REGID to directly access the gc_SetEvtMsk API function with the GCACT_ADDMSK parameter.

There are more than 300 such technology-specific access functions for telephony devices and more for TTS, voice recognition, and wave media features. Any telephony operation that has not been abstracted to the high-level API can be implemented as direct access functions.

In addition, hardware parameters can be set at startup. These execute board-level API calls. Arguments and results from these operations are written to a runtime log file to help configuration troubleshooting.

Application Development Language (ADL)

Beyond controlling the telephony devices, a computer telephony (CT) application confronts several more unique challenges.

- Multiple calls must be managed at one time. A multitasking architecture is needed.
- Each call might need to have a different conversation and work result. Program selection at call time is needed.

Applications typically run for days or weeks. Designs must be robust (effective error recovery, resource tight).

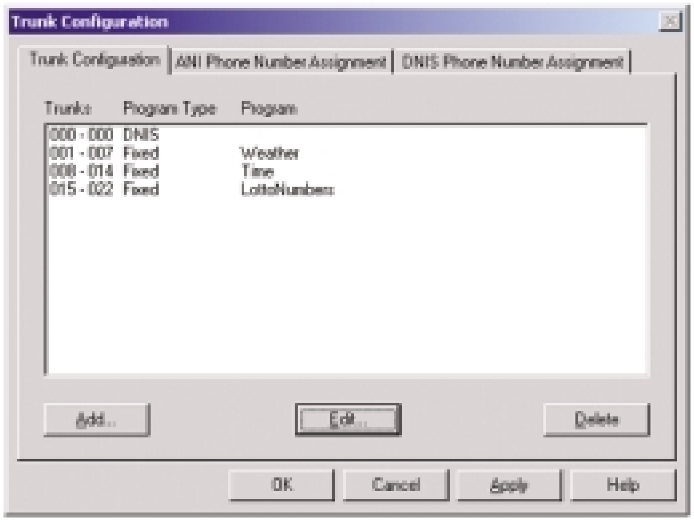
- Installations of hardware and software are often housed remotely, so hands-on operation is not always possible.
- Phone call processing is a real-time activity demanding real-time troubleshooting tools.
- Intel has addressed these requirements and more in its telephony-specific language, ADL. The sections that follow explain some of the facilities available to support this complex programming environment.

Task Management

ADL is multitasking, which means that two or more tasks may be executing at the same time. Each task has its own private copy of variables and arrays and its own independent path of execution, even if it is running the same program as one or more other tasks. In a multitasking environment, tasks usually continue their execution independently of each other. Sometimes, however, it is necessary for tasks to coordinate their activity or to communicate information. This is done with semaphores, messages, and global variables.

Task/Trunk Configuration

The Trunk Configuration program lets you specify which of your project's applications to use for the trunk lines in your system. When you run an ADL project, the Trunk Configuration settings determine which applications to start.



Trunks	Program Type	Program
000 - 000	DNIS	
001 - 007	Fixed	Weather
008 - 014	Fixed	Time
015 - 022	Fixed	LottoNumbers

Figure 1: Assign same or different tasks to each trunk (channel) in the configuration

You can assign a trunk resource or set of trunk resources to any application in your project. You can also specify that resources should be assigned to an application depending on the ANI or DNIS information received for each incoming call on those lines.

Static Data

Data areas in ADL are fixed at compile time. The starting position of each variable is fixed within the data area of the program. This makes it more efficient for ADL to locate data when executing an instruction. By default, ADL will not dynamically allocate memory. This avoids the need for garbage collection and the possibility of resource leaks.

TCP/IP and DCOM Communications

ADL can exchange messages and data with other applications on a local- or wide-area network using companion TCP/IP or DCOM communication programs. These tools enable remote monitoring of tasks, network status, and application progress.

Also, ADL includes Windows Service features that facilitate attendant-free system recovery (reboot) at remote locations.

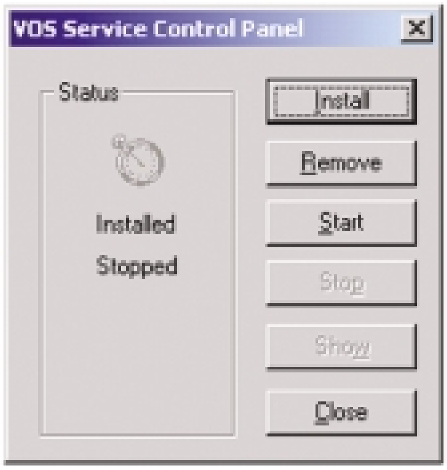
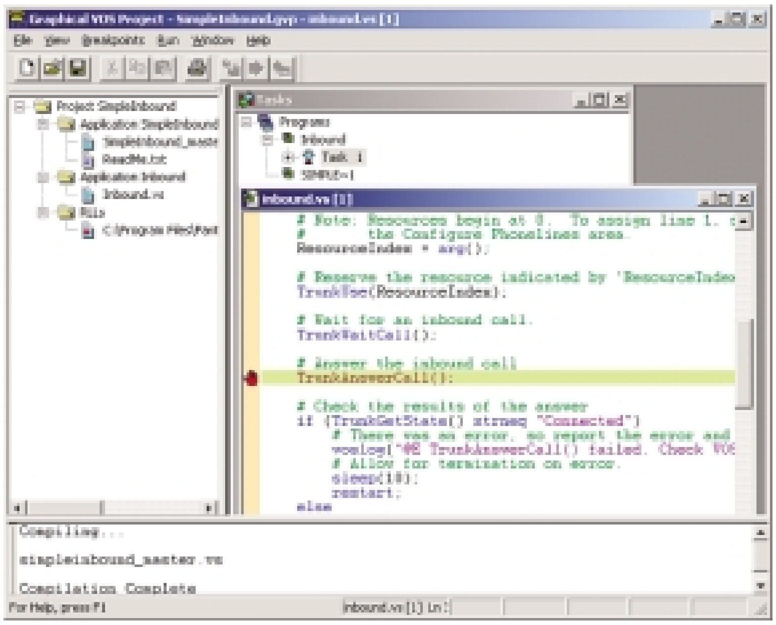


Figure 2: Windows Service installation and control utility

Symbolic and Real-Time Debugging

Different kinds of problems require different approaches. For locating design errors where the flow of control moves in unexpected directions, ADL comes with a symbolic debugger. Step from one instruction to the next, examine the content of variables, and set breakpoints — all common actions of today's advanced integrated development environments.



```
# Note: Resources begin at 0.  To assign line 1, c
#         the Configure Phonelines area.
ResourceIndex = arg();

# Reserve the resource indicated by 'ResourceIndex
TrunkUse(ResourceIndex);

# Wait for an inbound call.
TrunkWaitCall();

# Answer the inbound call
TrunkAnswerCall();

# Check the results of the answer
if (TrunkGetState() strneq "Connected")
    # There was an error, so report the error and
    vowlog("WE TrunkAnswerCall() failed. Check VOS
    # Allow for termination on error.
    sleep(10);
    restart;
else
```

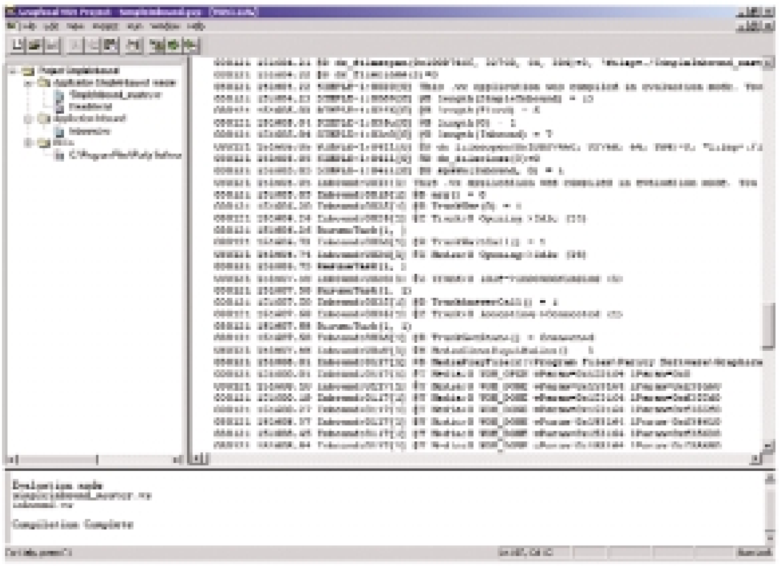
```
Compiling...

simpleinbound_master.vs

Compilation Complete
```

Figure 3: Symbolic debugger being used to set a breakpoint in one task

For examining what happened when a caller hung up at 3 a.m. the previous morning, ADL writes a runtime log file that can contain summary or detail traces of every action taken during execution. ADL, Resource Manager, and R4 events, states, and results can be captured to one-hundredth of a second.



```
[illegible]
```

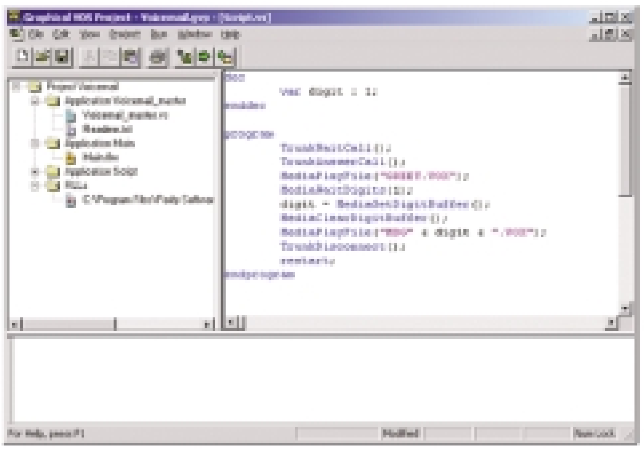
```
Evaluation mode
simpleinbound_master.vs
inbound.vs

Compilation Complete
```

Figure 4: Runtime log file showing events over time

Procedural Programming

ADL is a language similar to C or Basic with functions and variables.



```
dec
        var digit : 1;
enddec

program
        TrunkWaitCall();
        TrunkAnswerCall();
        MediaPlayFile("GREET.VOX");
        MediaWaitDigits(1);
        digit = MediaGetDigitBuffer();
        MediaClearDigitBuffer();
        MediaPlayFile("MSG" & digit & ".VOX");
        TrunkDisconnect();
        restart;
endprogram
```

Figure 5: ADL program that waits for a call, answers it, plays a prompt, gets DTMF touchtones from caller, and finally plays a specific file based on the digit collected

Graphical Programming

ADL Flowcharter, part of the ADL Studio, is a graphical tool that lets you create and edit telephony applications. Instead of writing source code, you draw telephony applications in the Flowcharter by inserting cells into a chart.

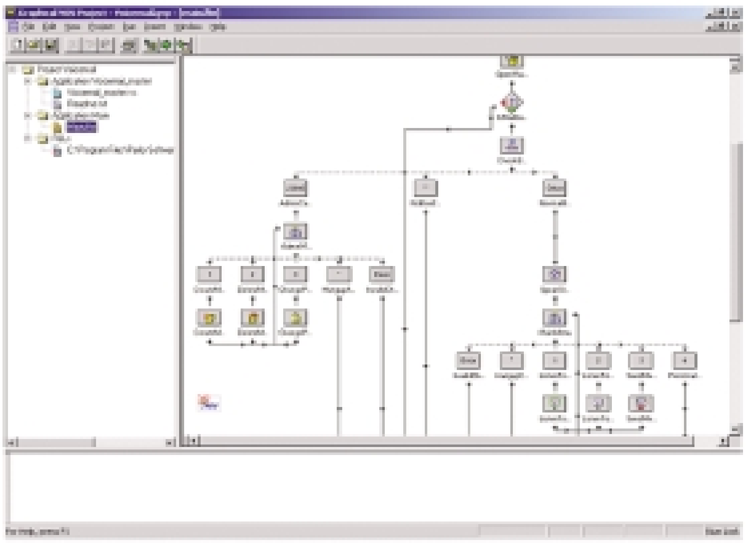
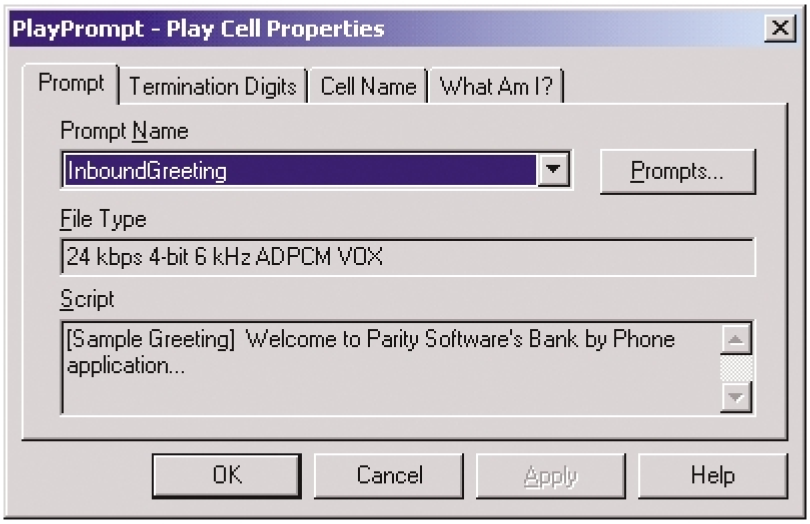


Figure 6: The cells of the chart define what actions should be executed in what sequence under what conditions

The cells are predefined ADL library functions that use arguments provided as properties in the cell.



Prompt Name: InboundGreeting

File Type: 24 kbps 4-bit 6 kHz ADPCM VOX

Script: [Sample Greeting] Welcome to Parity Software's Bank by Phone application...

Figure 7: Property dialog used to set the prompt name and terminating digit for the PlayPrompt cell

Performance and Density**

To achieve its goal of API transparency, CT ADE inserts a resource management layer that unavoidably introduces some overhead. However, that overhead has been minimized through optimized program design. But by far the most significant aspect of performance is how the application manages multiple channels — not how fast it attends to a single stream of instructions.

In any application, CPU usage varies according to the program design:

- Multiple exe (one channel per application) — This carries the heaviest CPU overhead because Windows must change from one process (program) to another to request and service each telephony state change.
- Multiple thread (one channel per thread in one process) — This design achieves better CPU performance but still needs the operating system to switch threads to manage the channels.
- Single thread state machine (all channels use one thread) — State changes are immediately known to the thread and can be handled directly without asking the operating system for help. Current channel state data is stored, next channel state data is retrieved, and processing continues.

Unfortunately, the greater the efficiency in CPU usage, the more difficult it is to design and implement. To build a complete, robust, and flexible state machine requires many years of effort. This is why Intel offers ADL, its implementation of a state machine for the control of hundreds of channels in a single thread.

Intel Benchmark Results**

Table 1 shows some actual performance results in both lab tests and the field.

Configuration	Applications Running on One PC	Density (Ports)	Platform	Memory	CPU
Windows 2000 Server Pentium® III processor (650 MHz)	Caller: Makes calls, disconnects, and repeats				
250 MB RAM Intel® Dialogic® DMV960A4T1 multifunction board Intel® Dialogic® System Release 5.1.1 CT ADE v8.2 SP1	Called party: Answers calls, plays a prompt (streams audio), disconnects, and repeats	288 (144 in, 144 out)	ADL	33MB	15%

Table 1: Intel benchmark results

This test used 15% of the available CPU cycles to perform all of the API calls, event handling, and state change housekeeping. The rest is available for other uses such as business logic or speech synthesis or recognition.

Customer ADL applications

The real-world results in table 2 show the performance of ADL applications that do have the important business functions integrated. Again, even if these applications are less complex than those you plan to build, it will not be the telephony abstraction work that limits the density per chassis.

Site	Processor Speed	Density	Application Type	CPU
Miami	500MHz	16 T-1s (384 lines)	Debit card	30%
Vancouver	1.26GHz	20 T-1s + 4 (484 lines)	Chat line	40%
Portland	650MHz	16 T-1s (384 line)	Debit card	50%
Israel	500MHz	16 E-1s (480 lines)	Debit card	30%

Table 2: Performance of ADL applications without business functions integrated

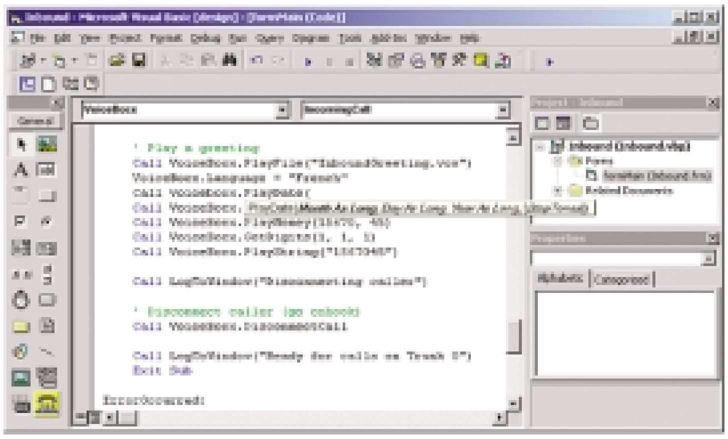
VOS Migration

ADL is the descendent of VOS* (Voice Operating System) from Parity Software. ADL introduces a complete new set of Resource Manager functions to VOS. The old-style functions *(sc_, DTI_, GC_…)*, referred to as VOS Legacy, will continue to be supported for backwards compatibility. However, for new projects Intel strongly recommends moving to the Resource Manager API.

Application Development ActiveX Objects (ADX)

Component Object Model (COM) controls, also referred to as ActiveX objects, contain useful services that can be easily incorporated as components into larger mission applications. This method of delivering complex system functionality within business-oriented applications has become very popular, especially with developers who prefer using graphical programming environments like Visual Basic or Delphi.

All of the telephony services provided by the Resource Manager described above are packaged into building blocks that expose their functionality during development (show their methods, arguments and properties), register that functionality with the operating system, and create the necessary objects at execution time.



```
VoiceBoxx                          IncomingCall

    ' Play a greeting
    Call VoiceBoxx.PlayFile("InboundGreeting.vox")
    VoiceBoxx.Language = "French"
    Call VoiceBoxx.PlayDate(
    Call VoiceBoxx. PlayDate(Month As Long, Day As Long, Year As Long, [illegible])
    Call VoiceBoxx.PlayMoney(15470, 45)
    Call VoiceBoxx.GetDigits(1, 1, 1)
    Call VoiceBoxx.PlayString("1567345")

    Call LogToWindow("Disconnecting caller")

    ' Disconnect caller (go onhook)
    Call VoiceBoxx.DisconnectCall

    Call LogToWindow("Ready for calls on Trunk 0")
    Exit Sub

ErrorOccurred:
```

Figure 8: Visual Basic program using ADX Voice method PlayDate that prompts for arguments in compliance with COM standard

The suite of controls collectively known as ADX, are:

- **Voice** — Analog, digital, IP, and HMP call control platforms
- **Fax** — CP, VFX, and DM3 hardware
- **Conference** — MSI, DCB, and DM3 hardware
- **Text-to-Speech** — Enabling use of SpeechWorks* Speechify*, Nuance Vocalizer*, L&H Real Speak*, and SAPI-compliant products
- **Automated Speech Recognition** — Supporting products from SpeechWorks, Nuance, Philips, and Microsoft
- **Network Hub** — Thread-to-thread or process-to-process data exchange using TCP/IP or DCOM

Programming Environments

Any environment that supports COM controls will support the ADX objects. Languages Intel has tested include C++, Visual Basic, Delphi, C#, and VB.NET. Reports from the field indicate that CT ADE controls have also been successfully used in JavaScript* Web pages, Visual FoxPro*, and PowerBuilder*. There are also third-party products that enable other languages like Java* to use COM controls.

Performance and Density

Early COM objects were written using Microsoft Foundation Classes (MFC). These classes were designed for visual objects with extensive support for user interface dialogs and document processing. Adapting MFC to COM programming led to bloated code and COM components with an unnecessarily large overhead. Intel ADX objects use the Active Template Library instead of MFC. ATL is specifically designed for COM development resulting in efficient COM components with a small footprint.

Using the same benchmark design and configuration that we used for ADL, a C++ multithreaded program with ADX Voice control results in twice the CPU usage but still with 70% left for data and other logic processing. Furthermore, these single chassis results can be improved by using a dual-processor CPU.**

Configuration	Applications Running on One PC	Density (Ports)	Platform	Memory	CPU
Windows 2000 Server Pentium, III processor (650 MHz) 250 MB RAM Intel® Dialogic® DMV960A4T1 multifunction board Intel® Dialogic® System Release 5.1.1 CT ADE v8.2 SP1	Caller: Makes calls, disconnects, and repeats Called party: Answers calls, plays a prompt (streams audio), disconnects, and repeats	288 (144 in, 144 out)	ADL	43MB	30%

Table 3: Intel benchmark results

CallSuite Migration

ADX is the descendent of CallSuite controls from Parity Software: VoiceBocx*, FaxBocx*, SwitchBocx*, ChatterBocx*, MatchBocx*, and NetHub*. Although with version 8.3 the names have been changed in the literature, the control names, as well as their interface signatures, are unchanged.

New Directions

What else can Intel do for the CT development community? Here are some of the alternatives Intel has been considering:

- **VoiceXML/SALT** — Automated voice services can be expressed using these markup languages, facilitating links to Web services from telephones. You can build automated voice services using exactly the same technology you use to create visual Web sites. For the Intel view of the current state of VoiceXML platforms, see “VoiceXML Platforms,” an Intel white paper. Read about the collaboration between Intel and Microsoft on an implementation of SALT at http://www.intel.com/home/trends/future/salt.htm and “Microsoft .NET Speech Technologies” at http://www.microsoft.com.

- **Windows library (not COM)** — Although Intel has implemented the most efficient form of COM objects, there are some program designs and host languages that could be better supported by a library of C++ telephony classes.
- **Linux*** — All the advantages of the Resource Manager, the telephony classes, and the ADL framework.

Let us know how these do or do not line up with your requirements for the future. Contact your Intel distributor (http://www.intel.com/reseller/index.htm) or the Intel product manager for CT ADE, Lyle Cowen (Lyle.Cowen@Intel.com; (415) 332-5656, x1310).

To learn more, visit our site on the World Wide Web at **http://www.intel.com**

1515 Route Ten
Parsippany, NJ 07054
Phone: 1-973-993-3000
Fax: 1-973-993-3093

Printed in the USA   Copyright © 2003 Intel Corporation All rights reserved.   Printed on recycled paper.   03/03   00-8541-001