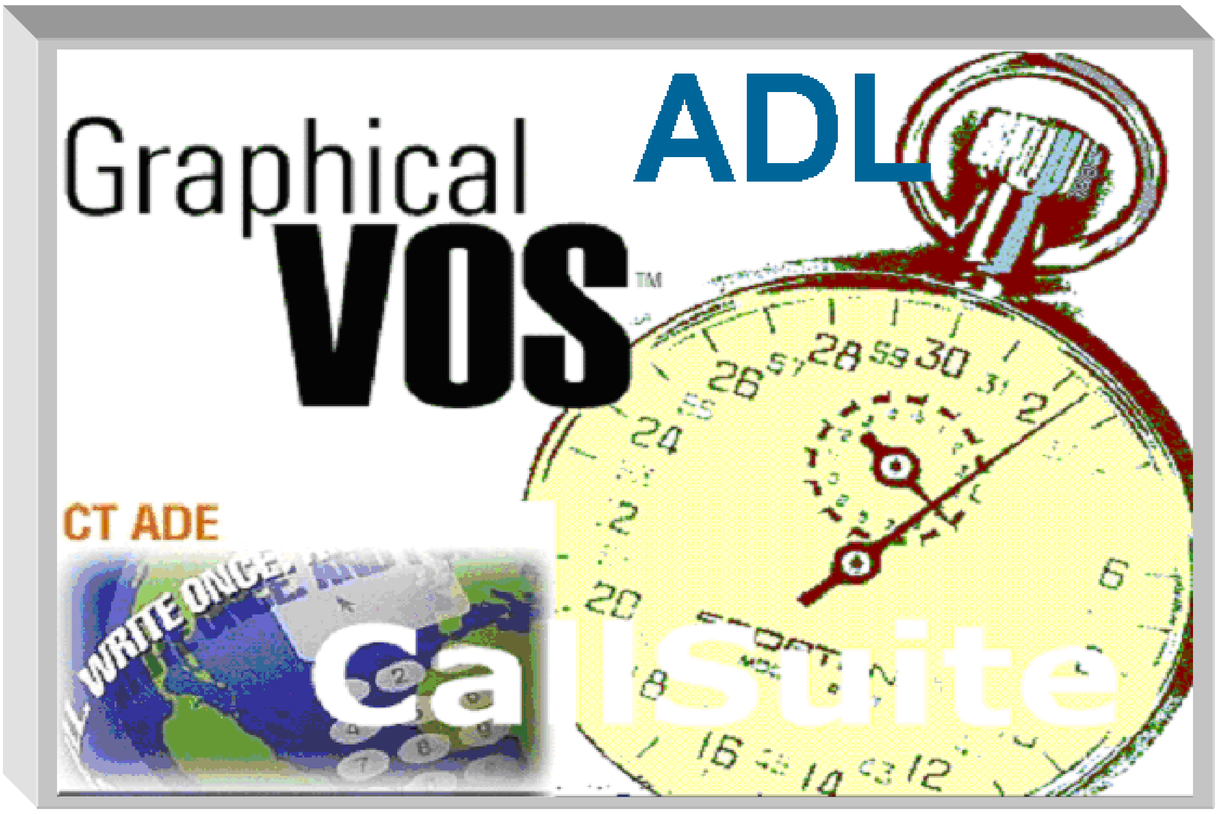


# Intel CT ADE
# TRAINING COURSE
# GVOS 8.2 SP1 (ADL)& CT ADE Architecture

**Three Harbor Drive**
**Suite 110**
**Sausalito, CA 94965**
**Tel.: (415) 332-5656**
**Fax: (415) 332-5657**

***http://www.intel.com/network/csp/products/ct_ade.htm***

# INDEX

# 1. Intel CT ADE

The core of any CT application is *The voice card.* A voice card is just a PC extension like an Ethernet LAN Adaptor or a Sound Blaster card, but with voice processing features. There are several types of PC buses: ISA (16 bit, 8 Mhz) and PCI (33 MHz). There are two kinds of network interfaces: Analog and Digital. The first one normally has sockets RJ-11 and the second one normally are E-1 (T1 in EE.UU.) or ISDN conexions (PRI or BRI).

Another important concept is the voice bus, which allows you to share the resources. Voice cards come with a special programming API based on Dinamyc Link libraries (DLL) for C/C++ environments. C/C++ language is normally used in complex low level application development (S.O. , device drivers, etc.). Documenting and maintaining C/C++ code is not trivial.

Intel's CT ADE primary objective is to help developers shortern development process thanks to an easy to use, full-featured, industrial-strength software development environment.

# 2. Introduction

CT ADE - Graphical VOS 8.2 SP1, is the lastest version of the Intel telephony tools. These tools include FlowCharter, VOS Source code editor, VOS Source compiler, Runtime, VOS Source code step by step debugger and Simphone: a telephone line simulator that uses any windows sound device (like sound blaster, for example). All these tools are now integrated in a single development environment: CT ADE.

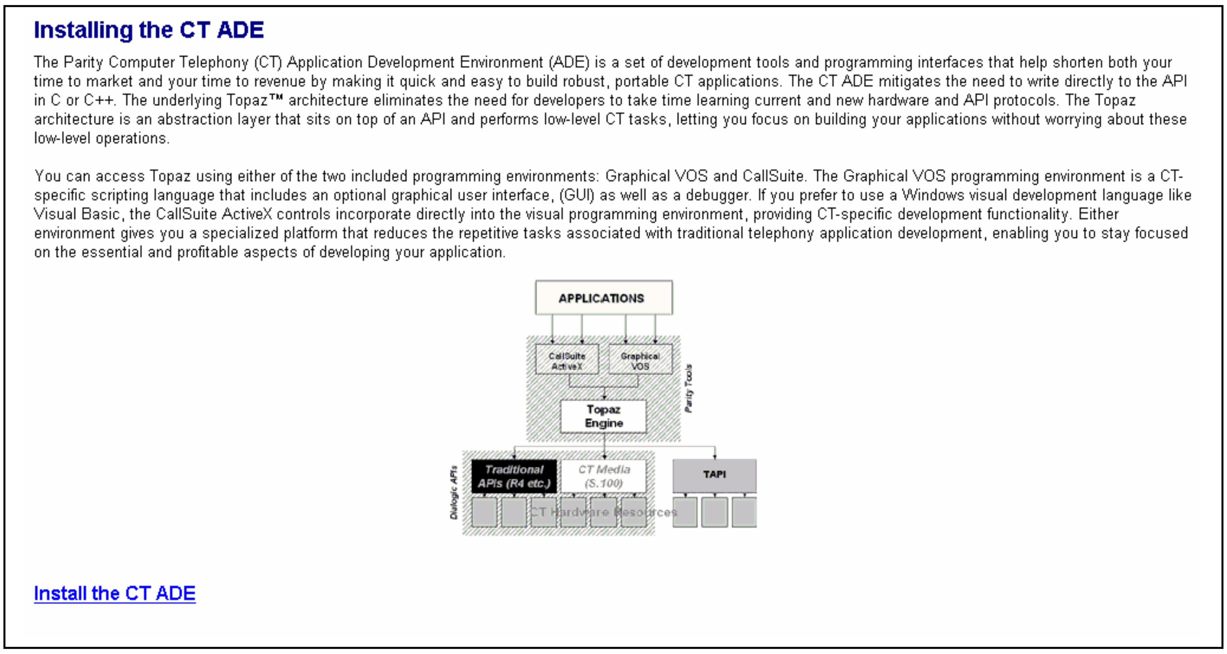
## 2.1. Intel Brand names

Since the acquisition of Intel over Parity Software, the product has been changing and adapting to the new times. Please take a look in the following table to recognize the new names:

- **Graphical VOS** → Application Development Language **(ADL)**
- **CallSuite** →Aplication Development Activex Objects **(ADX)**

## 3. CTADE, Firsts steps: Installation

First we have to install the CT ADE, therefore we will introduce the CD (AutoRun CD) in our CD Drive. The following pop up menu will appear on your screen:



### Installing the CT ADE

The Parity 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 CT ADE mitigates the need to write directly to the API in C or C++. 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.

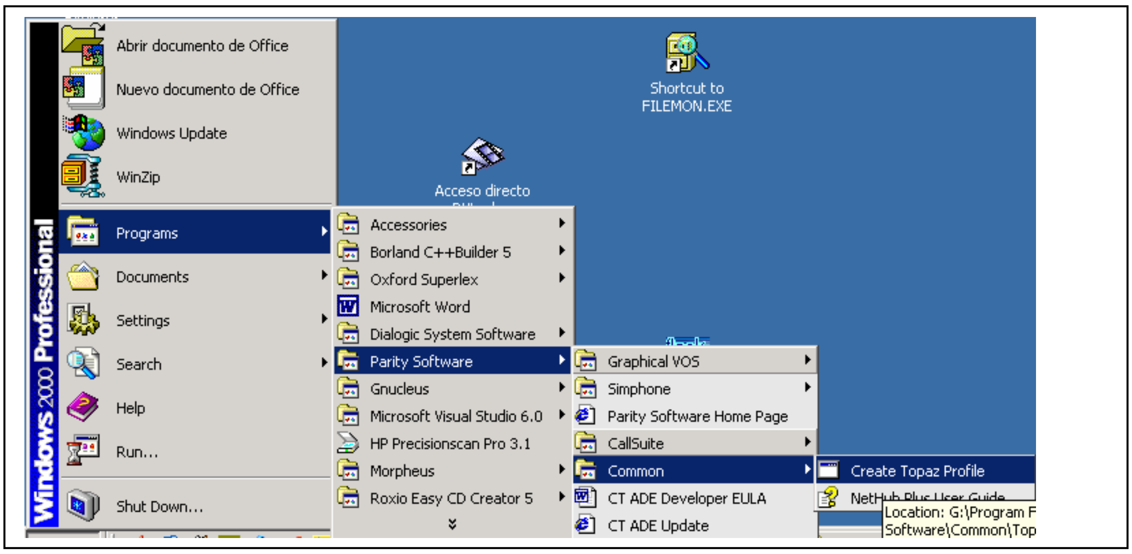
You can access Topaz using either of the two included programming environments: Graphical VOS and CallSuite. The Graphical VOS programming environment is a CT-specific scripting language that includes an optional graphical user interface, (GUI) as well as a debugger. If you prefer to use a Windows visual development language like Visual Basic, the CallSuite ActiveX controls incorporate directly into the visual programming environment, providing CT-specific development functionality. Either environment gives you a specialized platform that reduces the repetitive tasks associated with traditional telephony application development, enabling you to stay focused on the essential and profitable aspects of developing your application.

Install the CT ADE

We have just installed VOS with the CTADE Architecture engine.

## 4. CTADE, Graphical VOS

After installing Graphical VOS, we will see the following menu:



Now we have to configure the CTADE_A layer in order to detect which Dialogic drivers and boards are installed in the system (R4, S.100, etc…). CTADE_A is able to detect our dialogic configuration, to do so we will have to run the **TopazProfile exe**

Once we do that (only the first time and every time our dialogic configuration changes), we can use Graphical VOS with CTADE_A.

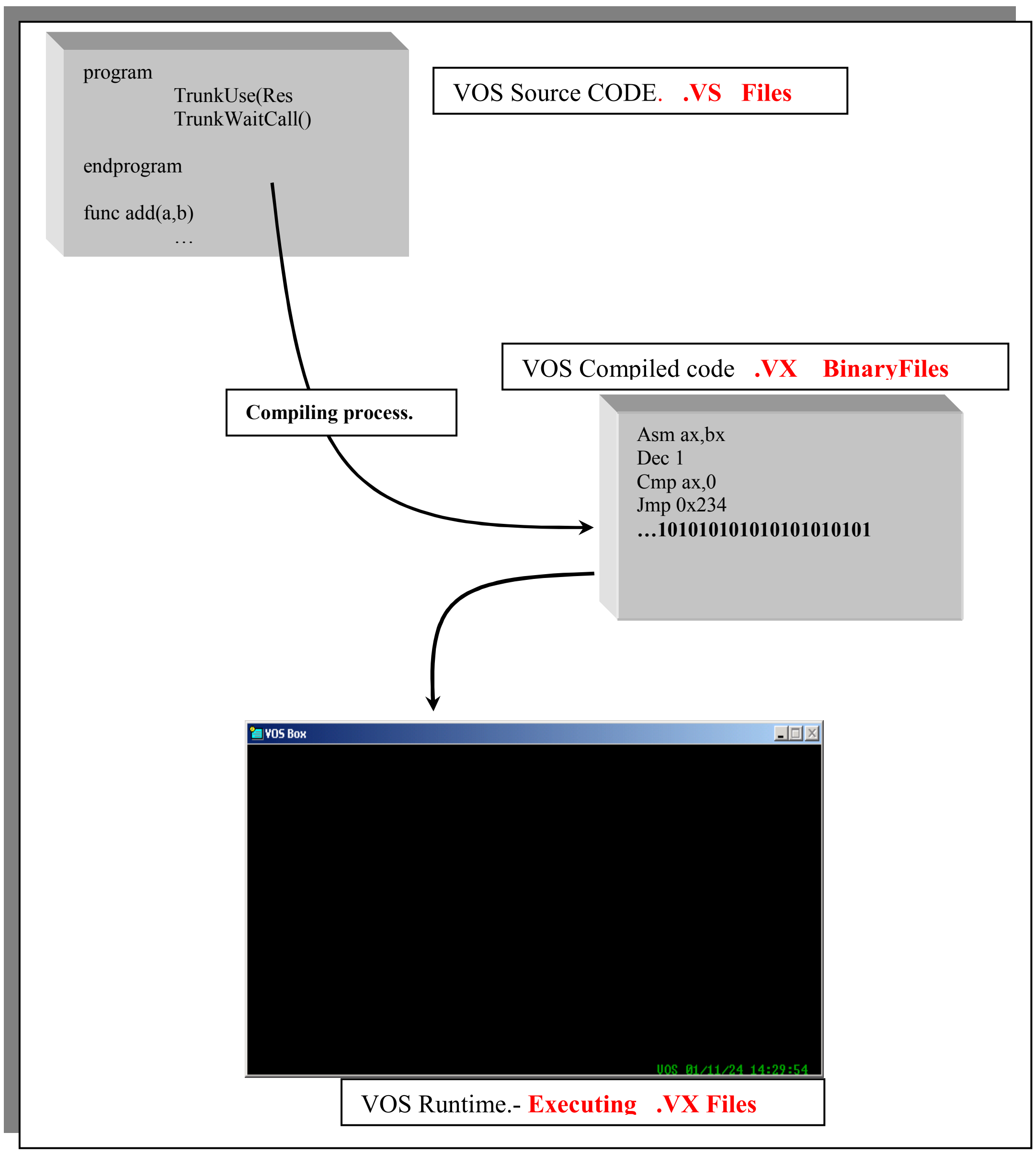
Now we are going to see some of the most important features of **Graphical VOS** app and how to obtain the best results from this tool.

# 5. VOS Language Reference

## 5.1. What is VOS?

VOS (Voice Operative System) is a:

- Script **Language**
- VOS language **Compiler**
- VOS Code **Runtime**



## 5.2. First VOS program

Let’s see a sample VOS program:

```
program
        vid_write("I am a VOS program!");
        vid_write("Type any key to exit...");
        kb_get();
        vid_write("Exiting now.");
endprogram
```

To demonstrate multi-tasking capabilities of VOS, we will show two programs running in parallel.

The following program shows a simple clock on the screen, updated once per second:

```
program
        for (;;)
                vid_cur_pos(0, 50);
                vid_print(time() & " " & date());
                sleep(10); # Wait one second
        endfor
endprogram
```

Now let’s do a third one to execute both programs simultaneusly:

```
program

        spawn(“clock”);

        spawn(“hello”);

endprogram
```

# 6. VOS Language

In this chapter we're going to review briefly the most important concepts of the VOS Language.

The VOS language is easy to learn, robust and fast. It is specially designed for call processing -- for example, there is no dynamic memory allocation, so live systems cannot run out of memory. The VOS language is simpler than Visual Basic and much simpler than C or C++.

## 6.1. Source Code

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

**Comments** may be included in source code by using the **# character**. (This is the so-called "pound" sign, which may appear as £ or another special character on non-US PCs; it is the ASCII character with code 35 decimal, 23 hex.) Comments continue up to the end of a line.

The end-of-line mark has no syntactical significance except that it terminates a comment. Multiple statements may be included in a single line, although this is discouraged because it generally makes the source code harder to read.

The source code for a VOS program is organized in the following way:

```
<variable declarations>

    program
            ...any number of statements...
    endprogram

    <onsignal declaration>
    <function declarations>
```

The **<declarations>** section informs VOS of the variables and constants to be used in the program. There may be any number of **dec..enddec** blocks, they may be positioned before or after the program..**endprogram** block or before or after any func..**endfunc** block.

The following is an example of a complete VOS program:

```
dec
    var line : 2;
    const MSG = "C:\MSG\message.vox";
enddec

program
    line = arg(1);
    if (line eq 0)
            line = 1;
    endif
    TrunkUse(line);
    TrunkWaitCall(line);
    TrunkAnswerCall(line);
    Message();
    restart;
endprogram
onhangup   # Hang-up processing
    TrunkDisconnect(),
    restart;
endonhangup

func Message()
    MediaPlayFile(MSG);
endfunc
```

## 6.2. 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"**

### *6.2.1. 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".** The double quotes may be omitted when writing a number, so "123" and 123 both represent the same string of three characters.

### *6.2.2. Logical values*

**True** and **False** are also represented as **strings**. False is represented as an empty string containing no characters, written "", and True is represented as "1" (a string containing the decimal digit 1).

## 6.3. Variables. Overview

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.

A variable has a maximum length. If a string longer than this length is assigned to the variable, it will be truncated to this maximum length. If you are running a Debug version of VOS, a warning message is issued if a string is truncated by assigning it to a variable. (If you want to avoid the warning, assign using a substr function.)

**Important**

**One of the most common programming mistakes that VOS beginners make is to forget that variables have fixed length and that values will be truncated. This is yet another good reason to make a habit of using the Debug version of VOS and to look at the vos1.log file on a regular basis to make sure that there are no error or warning messages.**

**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 why we chose to design a new language for VOS rather than using an existing language.

Variables must be declared (given a name and maximum length) before they can be used. Variables can be declared in two places: before the start of the main program, or inside of a function. An example program with a variable declaration block looks like this:

```
dec
        var x : 2;
        var y : 3;
enddec
program
        x = 12;
        y = 123;
        y = "Too long"; # y becomes "Too"
endprogram
```

Two variables are declared: one named x, with a maximum length of 2 characters, and one named y, with a maximum length of 3 characters.

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

Something to bear in mind is the Symbol Table concept. The VOS Symbol Table don't allow levels, this means that the functions names are globals and the variables names are visible in the task entirely:

```
dec
        var x : 2;
        var y : 3;
enddec
program
        x = 12;
        y = 123;
        y = "Too long"; # y becomes "Too"
endprogram
func suma(a,b)
dec
        var x:2; <<<< - ----- compilation error

enddec
        x = a+b;
        return x;
endfunc
```

## 6.4. Arrays

An array is a **set of stored strings** given a single name (identifier).

An array is one or more variables with the same length, all with the same name. The different variables in an array are distinguished by a number called the "subscript". For example, we could have an array called a with three variables in it, that could be named:

**a[1], a[2], a[3]**

Each of these three can be assigned values and used in other ways just like other variables. For example,

```
a[1] = "Hello";
x = a[3];
spk_num(a[2]);
```

Arrays may be declared wherever variables may be declared.

An example of an array declaration is:

```
dec
        var Arr[1..10] : 8;
enddec
```

```
Sum = 0;
for (Index = 1; Index <= 10; Index++)
        Sum += Arr[Index];
endfor
```

Any expression may be used for an array index.

In general, an array declaration is:

**<ArrayName> [ <MinIndex> .. <MaxIndex> ] : <MaxLen>**

An array index must be in the range 0 .. 255. This means that <MinIndex> and <MaxIndex> must be in the range 0..255.

There may be no more than 255 arrays defined in a single program.

For arrays bigger than that, I strongly recommend the use of **glb_dimx** for bigger arrays, of global dynamic memory allocation. We will see this later on.

## 6.5. Constants. Introduction

Constants are **fixed values** used by VOS. Constants are strings of characters. Strings are usually specified by typing the characters within double-quotes:

**"This is a string"**

If all the characters in the string are decimal digits, the double quotes may be omitted. Therefore, "123" and 123 specify the exact same string of three characters. Note that, because of this rule, -123 is therefore an expression with a unary minus operator followed by the constant string "123", whereas "-123" is a constant string with four characters.

Within a string, the backwards single quote character, `, has special significance. The ` and the following characters are interpreted as shown in the following table.

| ***Sequence*** | ***Puts this single character into the stored string*** |
|---|---|
| `` | Single backwards quote ` |
| `q | Double-quote " |
| `r | Carriage-return (hex 0A) |
| `n | New-line (hex 0D) |
| `t | Tab (hex 09) |
| `xx | Byte with hex value xx, eg `09 would have the same effect as `t. It is illegal to use `00. |

A name (identifier) may be assigned to a constant within a dec .. enddec block. Dec .. enddec blocks may be placed before the main program, in which case the variables declared may be used throughout the entire program, or following a func statement, in which case the constant name may be used only within the function. All variables are set to an empty string when VOS starts the program or when a restart statement is executed.

A typical declaration block looks like this:

```
dec
        var <name> : <length>, <name> : <length> ... ;
... more variable declarations ...
    const <name> = <value>, <name> = <value> ... ;
... more constant declarations ...
    enddec
```

For example,

```
dec
        const MAX_LINES = 24;
enddec
```

declares one constant named MAX_LINES which stands for the two-character string "24". Following this declaration, MAX_LINES may be used wherever the string "24" is desired within the source code.

The value assigned to a constant may be any constant expression. A constant expression is an expression where all values are constant strings and where operators and parentheses (but not function calls) are permitted. For example,

```
const Size = (1234 + 777)/45;

const Str = "ABC" & "DEF";
```

Constants that are passed using the -D command-line option to Vlc may also be used in constant expressions.

For another example of a named constant,

```
const LOCAL_AREA_CODE = 415;
```

With this definition, you can use LOCAL_AREA_CODE anywhere in the program you want the characters 415 to appear. This has two advantages. Firstly, when you change the program, perhaps to run in New York, you only have to make one change:

```
const LOCAL_AREA_CODE = 201;
```

By convention, constants are often named using UPPER CASE names, but there is nothing in VOS or vlc that expects or requires this convention.

## 6.6. Arithmetic Expressions.

In VOS you can use basic arithmetic expressions:

| Operation | Expression |
|---|---|
| Add | Left + Right. |
| Subtract | Left – Right |
| Multiplicacion | Left * Right |
| Divide | Left / Right |
| Change Sign | -Dcha. |

Operators have different "strengths," also called "precedences." 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. For example, in the expression (A+B)*C the addition will be performed first.

**Important**

**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 -231 to 231-1 can be represented, which is the range:

**-2147483648 to 2147483647**

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

**999999999 + 888888888**

will work correctly, but:

**(999999999*5)/4**

**will NOT** work because the intermediate result from the multiplication will overflow 32 bits.

## 6.7. Arithmetic with Decimal Point

At the time of writing, VOS has no built-in functions for dealing with fixed- or floating-point decimal arithmetic. The **FP RLL**, available at no charge for DOS and Windows, provides decimal arithmetic functions fp_add, fp_sub etc

## 6.8. Logical Conditions

VOS includes logical operations (and, or, not) and comparison operators (greater than, less than...) which give logical results (True or False). True is represented as "1", False is represented as "". Logical operators which combine two True / False values include

| Operación | Expresión |
|---|---|
| AND logical | Left And Right |
| OR logical | Left or Right. |
| NOT logical | not Right. |

Comparison operators

| Operación | Expresión |
|---|---|
| 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 |
| Equal as string | Left streq Right |

Samples:

**“001” eq “1”      True**

**“001” streq “1”   False !!!**

**“*” eq “#”        True !!!**

## 6.9. Assignments

The operator =  (assignment) is a special case of an operator. The left-hand side must be a variable (or array element). The right hand side is any expression. 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 most often used is +=, which could be described as "add to". For example,

```
n += 1;
```

adds 1 to n. 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 care about the result if all you want to do is add or subtract one from a variable. For example, the two alternative statements:

```
x++;
++x;
```

have exactly the same effect. The result matters if you use the result later in executing a statement, as in:

```
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 friends confusing, you can forget all about them; you can always achieve the same result using plain old =. If you're a C programmer and find them convenient, go ahead.

## 6.10. String Concatenation

The **&** operator combines two strings by placing the characters of the right hand side following the characters of the left hand side. This is called string concatenation. For example, "A" & "B" gives "AB". For another example:

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

This results in FullName being assigned **"Joe Smith".**

Important

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 for an example.

The expression:

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

will assign "0" to x! This is because & is stronger than *, so concatenation is done first, giving:

**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);**

## 6.11. Loops

A loop is a way to repeat one or more statements. The VOS language includes three types of loop: 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 syntax is as follows:

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

do
        statements
until ( test );

while ( test )
        statements
endwhile
```

The text shown in italics is replaced by appropriate code:

**statements**

One or more statements (which may themselves be loops) which are executed zero or more times as the loop repeats.

**initialize**

An expression which is executed once before the for loop starts. In the case of the for loop, this may be left blank if not required.

**test**

A logical condition which is evaluated each time through the loop and determines whether the loop continues executing. In the case of the for- and while-loop, it is evaluated before each loop, if True the loop continues to run. If test is False the first time through, the statements inside the loop are never executed. In the case of the do..until loop, the loop continues until the test is True. In the case of the for loop, the test may be left blank, in which case the value is always assumed to be True and the loop repeats for ever, or until exited by means of a goto, jump or return statement.

**increment**

An expression which is executed once at the end of each iteration through the for-loop. If the test is False the first time through the loop, the increment is never executed. In the case of a for loop, this may be left blank if not used.

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 n:

| | | |
|---|---|---|
| Sum = 0; | Sum = 0; | Sum = 0; |
| **for** (n = 1; n <= Max; n++) | n = 1; | n = 1; |
|     Sum = Sum + n; | **while** (n <= Max) | **do** |
| **endfor** |     Sum = Sum + n; |     Sum = Sum + n; |
| |     n++; |     n++; |
| | **endwhile** | **until** (n > Max); |

**A loop which repeats for ever** (or until exited by a **break**) 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.

## 6.12. Calling Functions

A function is a sequence of statements which has a name and produces a value. A function can have one or more "arguments," also called "parameters." Arguments are values which are copied into the function, inside the function arguments are referred to by names which behave very like variables except that they may not be assigned values.

There are three types of functions in VOS:

- **Built-in functions**. These are functions which are "hard-coded" into VOS and vlc, and are therefore always available to be used in a program.
- **User-defined functions**. These are functions which are written in VOS source code.
- **RLL functions**. These are functions which are written in C or C++. They are loaded from binary files called Runtime Link Libraries (RLL). On DOS, an RLL is a TSR. On Windows, an RLL is a DLL.

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

```
**vid_write**("Hello");          # Built-in function
**MyFunc**();          # User-defined
code = **Fquery**(queue, index);        # **RLL**
```

Since the syntax is exactly the same, you can't tell from the call to the function whether it is built-in, user-defined or in an RLL.

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 (the "return value") is obtained. The name of the function its list of arguments is thrown away and replaced by the return value.

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

```
vid_write("Hello"); # Return value not used
```

then VOS simply ignores the return value. If a useful value is returned, it may be used in an expression. For example, suppose that 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 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.

## 6.13. Functions and Library Files.

There are several ways to manage user functions in VOS: Functions files and library functions files.

Functions files are normally text files with the same name of the function that implements and with the extensión .FUN. Example:

File → add.fun

```
func add(arg1,arg2)
        return arg1+arg2;
endfunc
```

Library functions files are files with .vl extension. This files groups a collection of file functions (.FUN). Using mkvl command you can create this libraries with or without encryption option.

**mkvl <fich. Definition> <file lib>**

The definition file es a text file with references to .FUN files

```
func1.fun
func2.fun
\PROJ\FUNCS\LongFunctionName.fun
E:\SHAREDF\PROJ\OtherLongName.fun
```

## 6.14. Include Files

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:

```
dec
    include "project.inc"
    var MyVar : 2;
enddec
```

## 6.15. Internal Functions

VOS includes an extensive range of built-in functions. In this section we will give a brief overview of some of the important groups of functions:

### *6.15.1. Database*

The **db_** functions support database and index access using dBase-compatible DBF database files and **Clipper-compatible NTX indexes**. Databases can be opened and closed on the fly, one task may have several databases open at one time. Multi-user file and record locking is supported. Other database types are supported through RLLs, the details vary by operating system.

We recommend to use our Database Access RLLs , like **AdoRll** for any systems. With AdoRLL we can use SQL sentences in order to update /retrieve information to/from the Database Server (**Informix, Oracle, SQL Server, etc**.)

### *6.15.2. Serial Communication*

The **ser_ functions** support **RS-232 serial** communication through ports COM1 to COM4

### *6.15.3. Task Management*

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

### *6.15.4. 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).**

**This is covered later on the manual.**

### *6.15.5. String Manipulation*

There is an extensive set of functions for manipulating strings, including length, substr and many others. See String Functions for more information.

### *6.15.6. Screen and Keyboard*

The **vid_ and kb_ functions** provide for simple screen output and user input to a VOS program. Under Windows, you will probably want to create a graphical user interface--this is best done using the NetHub ActiveX to create the interface using Visual Basic, Delphi, Visual C++ or other Windows visual tool.

## 6.16. External Functions

All the VOS external functions are developed in C/C++. These functions are DLLs with some special entrypoints to inform VOS about the internal functions inside this DLL. We use to call these **DLLs** like **RLLs (Runtime Link Libraries).** We provide a **C/C++ developers kit** that allows you to create your own RLLs using any **C/C++ Windows Compiler (Visual C++, Borland C++ Builder, etc…).**

Below you'll see a list of the most common used RLLs:

### *6.16.1. NetHub Plus RLL.*

**NetHub Plus** is an enterprise component which supports communication between threads in a single process, processes in a single PC and/or nodes on a local or wide-area network.

NetHub Plus is available both as an **ActiveX control** and as a **VOS RLL.**

An instance of NetHub Plus can communicate with one or more other instances of NetHub Plus, either on one PC or across a network.

The VOS RLL loads a separate instance of NetHub Plus ActiveX for each VOS task.

**Messages**

Each instance of NetHub Plus can send and receive messages to and from any instance of NetHub Plus. Messages, which are strings of up to 127 characters, can be received with the MsgGet method or via the MessageReceived event.

**Semaphores**

Semaphores can be used to protect shared resources and critical sections of code. A semaphore is a special kind of variable that has two possible values: set or cleared. If a NetHub Plus instance sets a semaphore, no other instance may set it until it is cleared. A semaphore may only be cleared by the instance that set it.

**Shared NetVariables**

Shared NetVariables contain data (in variable-length strings) that is accessible to any instance of NetHub Plus on the network. NetVariables are identified by a name and an index number. The index number is similar to an array subscript. The maximum length of a shared NetVariable string is currently 127 characters.

Semaphores and NetVariables are identified by names. A name is a string of 1 to 127 characters. The first character must be a letter or underscore, the remaining characters may be letters, underscores or digits.

### *6.16.2. Ado RLL*

The **ADORll** is an RLL which provides direct access to Database servers via ActiveX Data Object (ADO) and Open Data Base Connectivity (ODBC, jet) respectively.

**We strongly recommend to use ADO Ole DB providers instead of ODBC providers because of stability and performance issues.**

### *6.16.3. Socket RLL*

The Sockets RLL makes it possible **to send and receive messages to VOS tasks running on remote PCs connected via an IP network**. The IP network could be an intranet, the Internet, a Novell network running TCP/IP or any other network which supports IP.

The RLL supports Windows NT only; it does not run under Windows 9x.

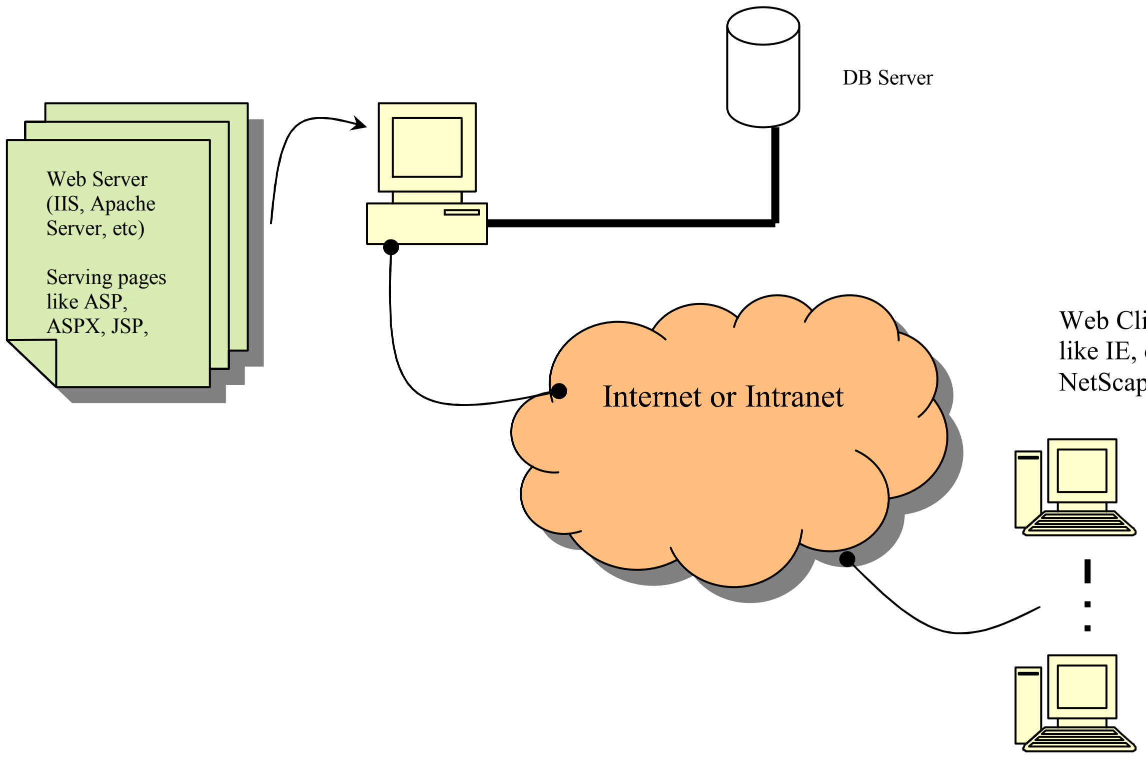
The Sockets RLL uses port number decimal 2222 (hex 8AE) by default. To change the connection port number, use ws_SetRemotePort.

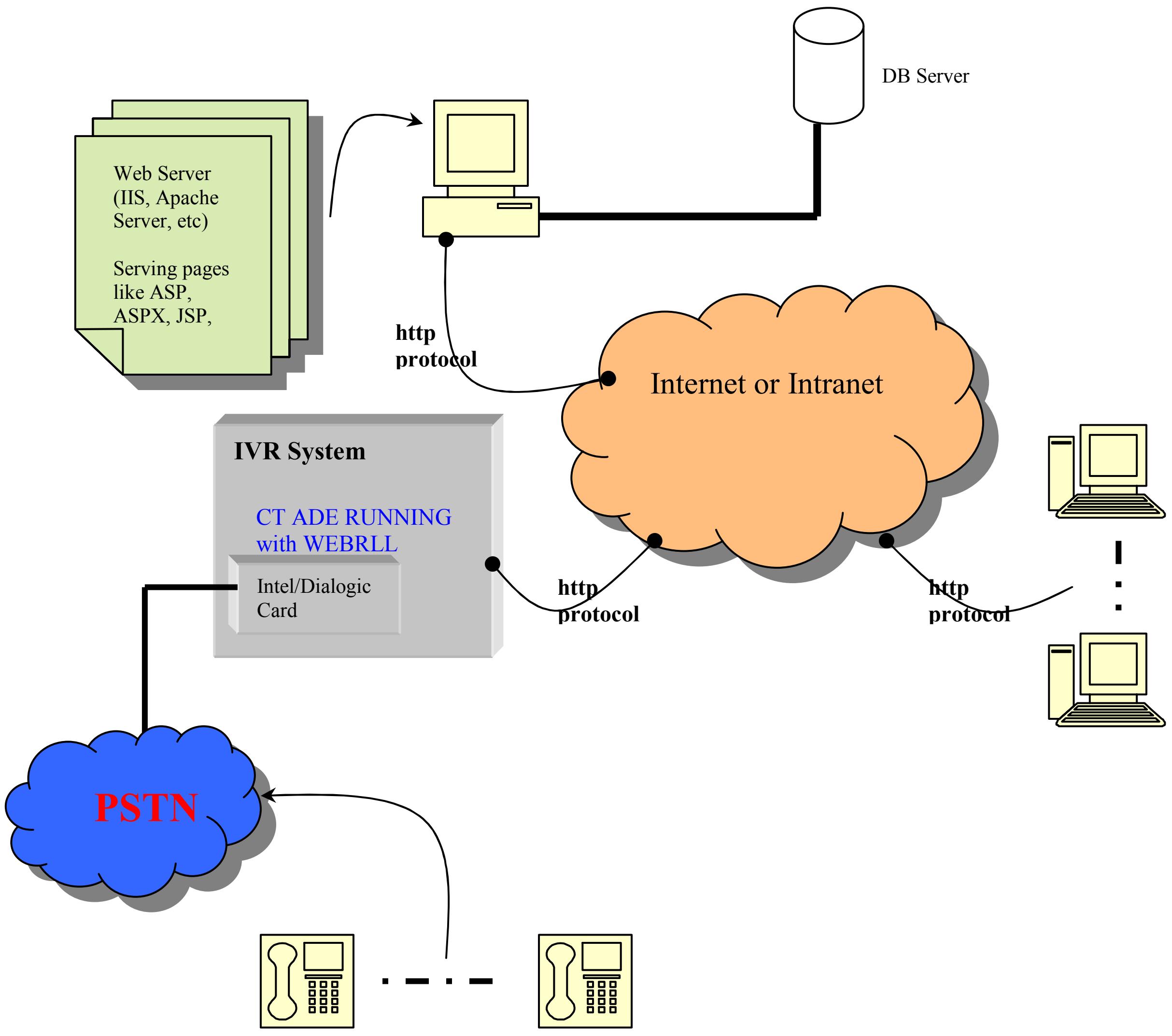
### *6.16.4. Web RLL*

WebRLL is a Runtime Link Library (RLL) for VOS.

WebRLL provides functions which enable your VOS program to:

- Get documents from a Web server
- Get data from active Web pages (e.g. from an e-commerce server)
- Send and receive files via FTP


DB Server
Web Server (IIS, Apache Server, etc)
Serving pages like ASP, ASPX, JSP,
Internet or Intranet
Web Clients like IE, or NetScape



## 6.17. Multi-tasking

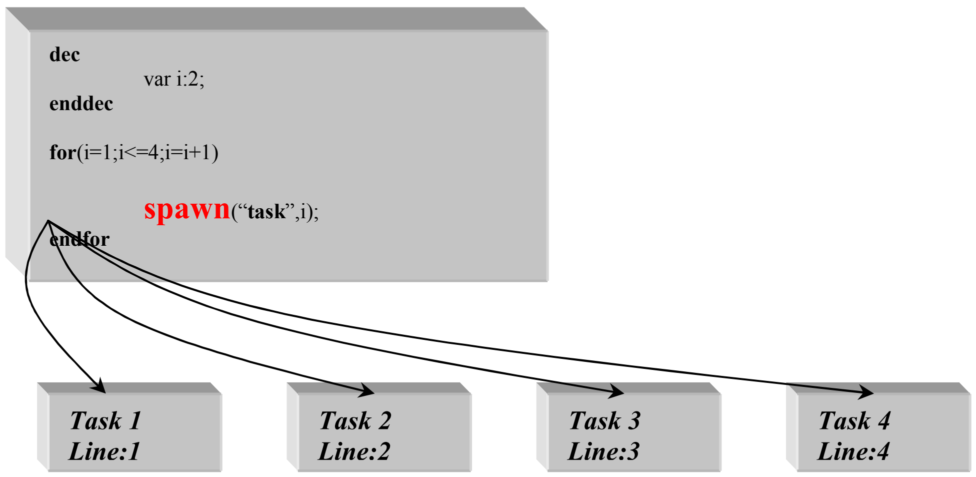
VOS is able to run a high number of lines simultaneously. VOS has its own Multithreading engine to manage each task. A simple VOS application for one channel:

```
dec
            Var linea:2;
enddec
program

linea = arg(1)
TrunkUse(linea);
MediaPlayFile(linea,”bienveni.wav”);
restart
end
```

***Task 1***

The previous figure shows a simple VOS task for only one channel. If we want to use this program to manage 4 lines, we will have to:



```
dec
        var i:2;
enddec
for(i=1;i<=4;i=i+1)
        spawn("task",i);
endfor
```

Now we have 4 tasks running inside VOS. From the beginning we had only one task that spawned 4 tasks passing one argument (the channel number) to each other. The function arg(1) retrieves the argument number 1. In that case, arg() is the channel number.

There are 2 ways to spawn VOS tasks: **synchronously and asynchronously**. To spawn tasks **asynchronously** we will use:

**task_nr = spawn(filename[, arg, arg...])**

To spawn tasks **synchronously** we will use:

**string = exec(program_name[, arg, arg...])**

## 6.18. Task Management.

As we saw in the previous chapters, VOS is able to manage several tasks simultaneously for multichannel applications or to perform “backgroud” tasks. Each VOS task has its own group of variables and arrays, and its own execution path.

There are some basic concepts to manage VOS tasks.

### *6.18.1. Task number.*

Each executing task has a unique task number, which may be 0, 1, ... The task number is sometimes called the process id or pid, a hang-over from UNIX terminology where a task is known as a process. The getpid function returns the task number of the current task, for example:

my_task_nr = **getpid**();

### *6.18.2. Suspending a Task*

The current task may be suspended at any given time by calling the sleep function. The argument is the length of time to wait before resuming execution, in tenths of a second. For example,

**sleep**(50);                    # Wait five seconds

The sleep period must be less than 12 hours, i.e. less than 432000.

A task which has been "put to sleep" takes no processor resources at all (VOS v5 or later). This means that performance problems which might be caused by polling loops can be alleviated by adding a short sleep to the loop. For example, if a task is waiting for a global variable to change value or for an incoming call:

```
while (glb_get(0) strneq "NEW" and not TrunkGetState() strneq "Idle"))
                              sleep(2);
endwhile
```

### *6.18.3. Killing a Task*

A task which is executing may be killed using the kill function, which takes a task number as an argument:

```
kill(condemned_task_nr);
```

The task will be killed immediately unless it has been suspended awaiting the completion of a built-in function such as sleep, sem_set, sc_play etc. In the case where the task has been suspended, it will be killed at the point where the built-in function currently in progress completes.

The return value from kill indicates the success of the operation:

| | |
|---|---|
| -1 | Invalid task number (this has never been an active task). |
| 0 | Task was suspended, has been scheduled to die when current function completes. |
| 1 | Invalid task number (task was once active but has now stopped). |

Task killed immediately.

### *6.18.4. Semaphores, messages and global variables*

In a multi-tasking environment like that provided by VOS, tasks usually continue their execution independently of each other. Sometimes, however, it is necessary for tasks to coordinate their activity or to communicate information.

There are four sets of functions in VOS which are provided for inter-task features:

- **Semaphores**
- **Messages**
- **Globales Variables**

Semaphores are usually used to manage what are called "critical sections" of code. A critical section is a sequence of code which may fail if two tasks are executing the same piece of code at the same (or close to the same) time.

| | |
|---|---|
| **sem_set(SemNr)** | **Sets the given semaphore number, waiting for it to clear if the semaphore has already been set by another task.** |
| **sem_test(SemNr)** | **Tests the semaphore and sets it if it's not already set (don't wait)** |
| **sem_clear(SemNr)** | **Clears the given semaphore, if it has been set by the current task.** |
| **sem_clrall** | **Clears all semaphores (if any) owned by the current task.** |

Sample

```
sem_set(1);

        # Update Critial-Zone

sem_clear(1);
```

There are 2 kinds of semaphones: Number semaphores or Named semaphores. The Named semaphores are semaphores that you can refers using a name. The following list show us the functions to manage named semaphores:

| | |
|---|---|
| **sem_setx(SemName)** | **Set semaphore, waiting until it clears if needed.** |
| **sem_testx(SemName)** | **Set semaphore only if it is clear (don't wait)..** |
| **sem_clearx(SemName)** | **Clear semaphore** |
| **sem_clrallx** | **Clears all semaphores owned by this task.** |

The Messages are sent between task using **msg_** functions. With these functions we are able to send and receive string messages among VOS tasks.

**code = msg_put(pid, string);**

**string = msg_get(seconds);**

**pid = msg_pid();**

**name = msg_sendername()**

**msg_flush()**

The Global Variables are very useful in order to share information between VOS tasks. The functions to use this Global Variables are as follow:

**Globales Variables**

**glb_get** value = glb_get(num)
Gets the current value of the given global variable number

**glb_set** glb_set(num, string)
Sets the global variable to the given string

**Named Globales Variables**

**glb_clrallx** glb_clrallx(Name)
Sets all values in the global to an empty string.

**glb_decx** glb_decx(Name[,Index1, ...])
Subtracts 1 from value in a Named global variable

**glb_dimx** glb_dimx(Name[,Lower1,Upper1, ...] [,MaxLength])
Creates a new Named global variable

**glb_getsizex** glb_getsizex(Name)
Returns the MaxLength for this global specified in the call to glb_dimx

**glb_getx** glb_getx(Name[,Index1, ...])
Get a value from a Named global variable.

**glb_incx** b_incx(Name[,Index1, ...])

Adds 1 to value in a Named global variable.

**glb_lowerx** lb_lowerx(Name[,DimNr])
Returns the lower limit on the index for the given dimension number as specified in the call to glb_dimx

**glb_nrdimsx** lb_nrdimsx(Name)
Returns the number of dimensions of the global variable. Returns 0 for a scalar, 1 for a vector, 2 for a matrix and so on.

**glb_rangex** lb_rangex(Name[,DimNr])
Gets the index range for the given dimension number (zero is assumed if not given)

**glb_setallx** b_setallx(Name, Value)
Sets all strings in the given global to the given value.

**glb_setx** b_setx(Name[,Index1, ...], Value)
Sets a string in the given global to the given value.

**glb_upperx** lb_upperx(Name[,DimNr])

Returns the upper limit on the index for the given dimension number as specified in the call to glb_dimx.

## 6.19. Date and Time Functions.

In IVR systems is very common to use dates and times for certains taks (for example, the duration of the call). The dates&time functions in VOS are as follow:

| FUNCION | SINTAXIS | DESCRIPCIÓN |
|---|---|---|
| date() | YYMMDD = date()<br><br>YYYYMMDD = date(1) | Returns the current machine date as a six-character string YYMMDD or as an eight-character string YYYYMMDD. |
| Dateadd() | New_Date=Dateadd(old_date,days) | Adds or subtracts a given number of days from a date, and returns the resulting date. |
| Datecvt() | Datecvt(fecha,tipo) | Converts a date from YYMMDD or YYYYMMDD format to indicate either the day of the week or the day number within the year (from 0 to 364 or 365).<br><br>0 Returns the day of the week |

| | | |
|---|---|---|
| | | 1 Returns the day of the year |
| Ticks() | Tick_count = ticks() | Returns the current BIOS tick count. |
| Timeadd() | YYMMDDHHMMSS=timeadd(fecha,hora,segundos) | Returns the date and time resulting from adding seconds to the given starting date and time. |
| Timesub() | Segundos=timesub(fecha1,hora1,fecha2,hora2) | Returns the difference in seconds between two dates (1 - 2) which are specified in YYMMDD, HHMMSS format. |
| Time() | HHMMSS=time() | Adds time to date. |
| Tmr_sec() | Segundos=tmr_sec() | Returns the number of seconds which has elapsed since the most recent call to tmr_start by the current task. |
| Tmr_start() | Tmr_start() | Starts a timer for the current task |

## 6.20. String functions

| | |
|---|---|
| ascii_code = **ctoi**(character) | Returns the ASCII value of the first character in the string argument |
| string = **itoc**(integer_value) | Converts an integer value to a string. The returned string contains one byte, the byte has the specified integer ASCII value |
| hexstr = **itox**(integer_value) | Converts the integer value to a hexadecimal value. |
| nrchars = **length**(string) | Returns the number of characters in its argument string. |
| newstring = **ljust**(string, char, width) | Left-justifies the string argument, padding on the right with the first character in the char argument to create a string of length width |
| newstring = **rjust**(string, char, width) | Right-justifies the string argument, padding on the left with the first character in the char argument to create a string of length width |
| comp = **strcmp**(string1, string2) | Returns a value based on an alphabetical ("lexical") comparison between the two strings. The comparison is done character by character using the ASCII codes for each character. The return value is > 0, 0 or < 0 depending on whether string1 is greater than, identical to or less than string2 using this method of comparison. |
| yes_or_no = **strcnt**(string1, string2) | Returns 1 (true) or an empty string (false) if string1 contains string2 as a substring |
| new_string = **strend**(string, count) | Returns the last count characters in string. If count specifies more characters than the length of the string, the whole string is returned. |
| index = **strindex**(str, str1, str2, str3 ...) | Returns the index 1, 2, 3 of the first string str1, str2 … to match str, or 0 if |

|  |  |
|---|---|
|  | the string is not found in the list. |
| new_string = **strltrim**(string) |  |
| new_string = **strltrim**(string, char) | Returns the string obtained by stripping leading characters from the string argument. If not specified, the char argument defaults to a single blank, " ". If char is given, the first character in char is stripped. |
| new_string = **strlwr**(string) | Converts any upper-case, English characters to lower case and returns the resulting string. |
| pos = **strpos**(string, substring) | Returns the position of the substring within the string, or zero if the substring is not found in the string |
| new_string = **strrtrim**(string) |  |
| new_string = **strrtrim**(string, char) | Returns the string obtained by stripping trailing characters from the string argument. If not specified, the char argument defaults to a single blank, " ". If char is given, the first character in char is stripped. |
| string = **strselect**(index, str1, str2, .... ) | Returns str1 if index is 1, str2 if index is 2, and so on. If index is < 1 or > number of string arguments given, then an empty string is returned. |
| new_string = **strstrip**(string[, char]) | Removes every occurrence of the given character within the string, and returns the resulting string. If no character is given, blank is assumed. The character is specified as a string of length one. |
| new_string = **strupr**(string) | Converts any lower-case, English characters to upper case and returns the resulting string. |
| substring = **substr**(string, from[, chars]) | Forms a substring of the first argument, starting at character position from and taking the next chars characters (defaults to the rest of the string if chars is not specified). Character positions are numbered from one |
| integer = **xtoi**(hex_value) | Converts a hexadecimal value to a decimal value |

## 6.21. File and Directory Functions (fil_, dir_)

VOS provides a number of built-in functions for reading, writing and searching files and directories. Any file which is "visible" to a command prompt, including network server files, can be accessed through these functions

| | |
|---|---|
| **find_end**() | Ends the directory search started by dir_first. Must be called following a dir_first before the current or any other task can begin a new directory search, even if dir_first returned an error. |
| filename = **find_first**(pathname, attributes) | Starts a directory search of the given path name, which may include drive and pathname components and which may include DOS wildcards (? and *) in the final name part of the path. |

The attributes argument includes one or more of the following characters indicating the attribute(s) of the files to be found:

| | |
|---|---|
| n | Normal file |
| r | Read-only file |
| h | Hidden file |
| s | System file |
| v | Volume label |
| d | Sub-directory |
| a | Archive bit set |

If successful, dir_first returns the file name (only the name.ext part, without any preceding directory path or drive). If the path or

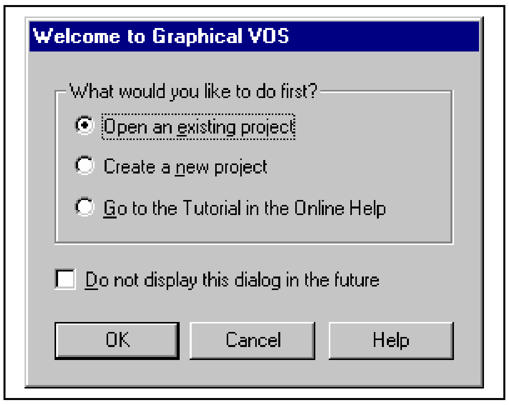
| | |
|---|---|
| | file given was not found, or if another task (or the current task) is currently performing a directory search, the return value is "?". To complete a directory search, dir_end must be called. |
| filename = **find_next**() | Returns the next matching file as given to dir_first, or "?" if no further files can be found or if no valid call has been made to dir_first. |
| free_bytes = **dos_diskfree**(drive) | Returns the number of bytes free on the given drive. The drive argument specifies the drive: zero means the current drive, 1 means drive A:, 2 means B:, 3 means C: and so on. |
| code = **dos_mkdir**(directory) | Attempts to create the given directory, which may be a path name and may include a drive letter. |
| code = **dos_rmdir**(directory) | Attempts to remove the given directory, which may be a path name and may include a drive letter. |
| code = **fil_close**(file) | Closes the given file, returns 0 if successful, a negative error code if the file handle file is not valid. |
| **fil_closeall**() | Closes all files (if any) opened by the current task. |
| code = **fil_copy**(source, target) | Copies source file to target file. Both source and target file names may include drive and/or path components. |
| code = **fil_delete**(name) | Attempts to delete the file with the given name. |
| bool = **fil_eof**(file) | Returns 1 if the file position is at or past end-of-file for the given file, 0 if the position is not past the end of the file, or a negative error code if file is not an open file handle. |
| line = **fil_getline**(file_handle) | Returns the line starting at the current file position, an empty |

| | |
|---|---|
| | string on error. |
| info = **fil_info**(name, infotype) | Returns information about the file with the given path name. Returns one of the following, determined by the info argument, or an empty string if the file does not exist. Options for infotype are: 1 File size; 2 Date YYMMDD; 3 Time HHMMSS; 4 Directory/file ("D" or "F"); 5 Read only? ("Y" or "N"). |
| code = **fil_lock**(File_handle, Position, Bytes) | Attempts to lock the given region of the file. |
| file = **fil_open**(name, mode) | Opens the file with the given path name, which may optionally include drive and directory components, with the given mode. |
| code = **fil_putline**(file_handle, line) | Attempts to write the string in the line argument to the end of the file, appending carriage-return and line-feed control bytes (DOS, Windows) or carriage-return byte (UNIX) to terminate the line |
| bytes_read = **fil_read**(file_handle, buf, bytes_to_read) | Attempts to read bytes_to_read bytes from the given file into the given buffer. |
| code = **fil_rename**(oldname, newname) | Renames the file with name oldname and gives it name newname. |
| pos = **fil_seek**(fil, pos, mode) | Moves the position of the given file according to the mode argument, which should be one of the following values |
| code = **fil_setsize**(file, bytes) | Sets the size of the file to the given number of bytes. If the number of bytes is less than the current file size, the file will be truncated. If the number of bytes is longer than the current file size, the file will be extended with "random" data (i.e., data currently residing in currently unassigned sectors). |

| | |
|---|---|
| code = **fil_unlock**(file, address) | Clears a lock in the file |
| bytes_written = **fil_write**(file, buf, bytes_to_write) | Attempts to write the given number of bytes from the given buffer to the given file. |
| code = **fil_writes**(file, string) | Attempts to write the given string argument at the current file position in the file. |

## 6.22. Graphical VOS, Introduction

When we first execute Graphical VOS we see this screen:


Welcome to Graphical VOS

What would you like to do first?

(●) Open an existing project

( ) Create a new project

( ) Go to the Tutorial in the Online Help

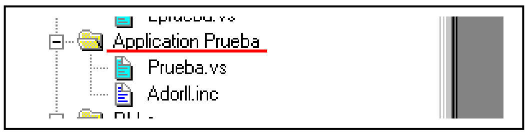
☐ Do not display this dialog in the future

OK  Cancel  Help

Here we can see a new concept called **PROJECT**. This is new for developers using previous VOS releases. The Project, like other programing tools in the market (Visual Basic, Visual C, Borland C++ Builder, etc..) contains all the necessary files to create the desired application. A VOS project includes:

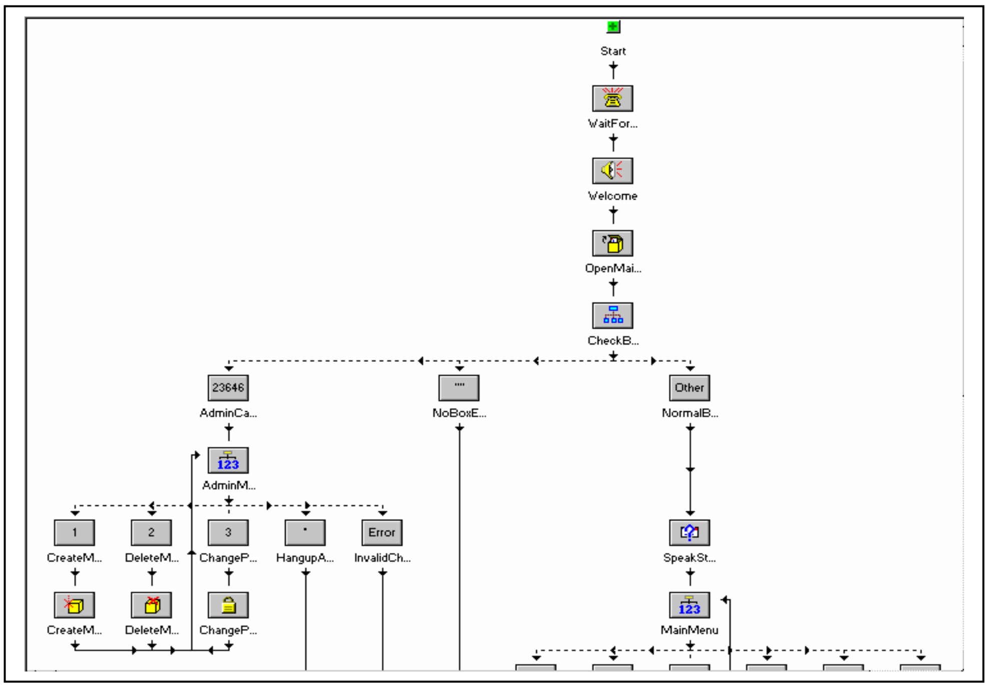
- VOS Source Code (.VS)
- Flow Charts (.flw)
- RLLs associated to the project.
- INC , FUN, DEC, VL Files.

The **VOS Source Code (.VS) files** are text files that contains source code in VOS language. Each VOS Source code file become part of the project in a folder called **Application** with the follow format: Application+ Vos Source Code name:



In every folder we can have references to include files (.inc), function files (.fun), declaration files and function library files (.vl).

The Flow charter (.flw) files refer to Flow Charting CTI applications. For example:



## 6.23. Graphical VOS. Firsts steps

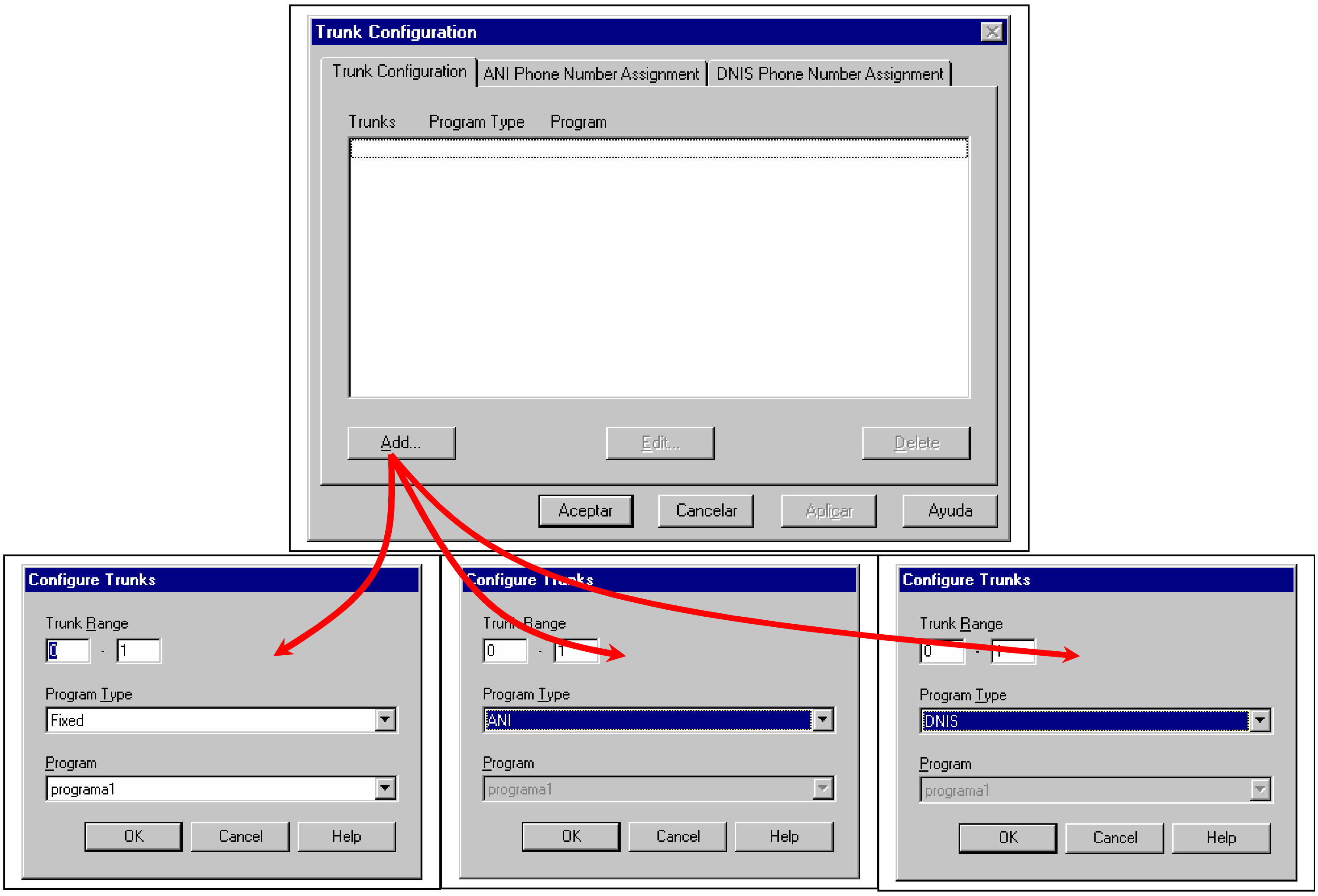
Now, let´s create a new **Vos project** to see the configuration and customisation process. After we create a new VOS Project, we will have a folder application called **master**.

**Very Important !!! Please, never remove the reference to the program master because it is required by VOS!**

Once we have reached this point, we can add/delete/edit Flow Charting and VOS applications using the **New** function from the File menu.

## 6.24. Graphical VOS. Basic Configuration.

Let's first configure the Trunks in the system (menu Project-> Configure Trunks).



Trunk Range: 0 - 1
Program Type: Fixed
Program: programa1

Trunk Range: 0 - 1
Program Type: ANI
Program: programa1

Trunk Range: 0 - 1
Program Type: DNIS
Program: programa1

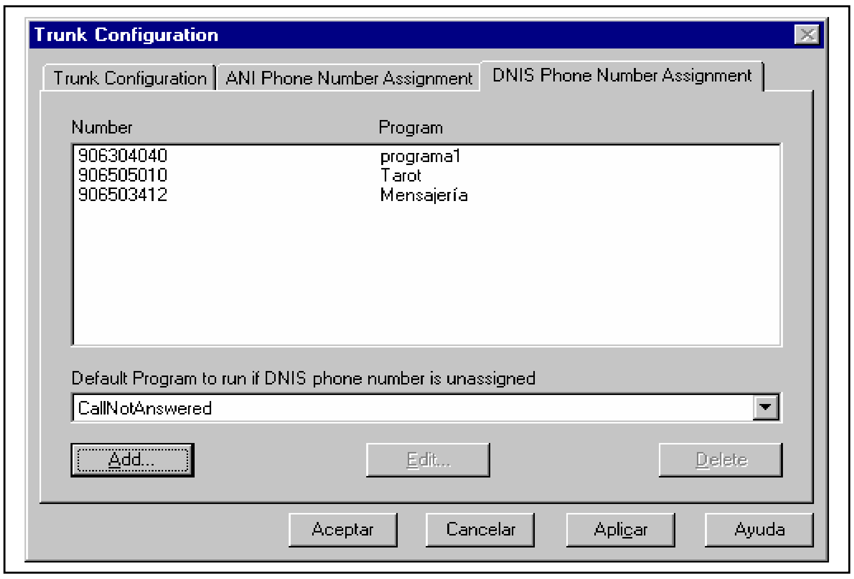
In the Main Trunk Configuration screen there is a Tab called Trunk Configuration. This tab is used to add an application for each channel as well as the type of the program associated to this channel. There are 3 types of programs:

1. **Fixed**: This is the default type program. Whenever an inbound call comes in through that channel, VOS will execute this program.

2. **ANI:** For the selected range of channels, VOS will execute the applications whose ANI appear in the tab "ANI Phone Number Assignment"

3. **DNIS:** For the selected range of channels, VOS will execute the applications whose DNIS appear in the tab “DNIS Phone Number Assignment”

Tab ANI and DNIS are used to control the application that will execute depending on the ANI or DNIS received. A typical example is 900 audiotext systems with different services in differents DNIS.



| Number | Program |
|---|---|
| 906304040 | programa1 |
| 906505010 | Tarot |
| 906503412 | Mensajería |

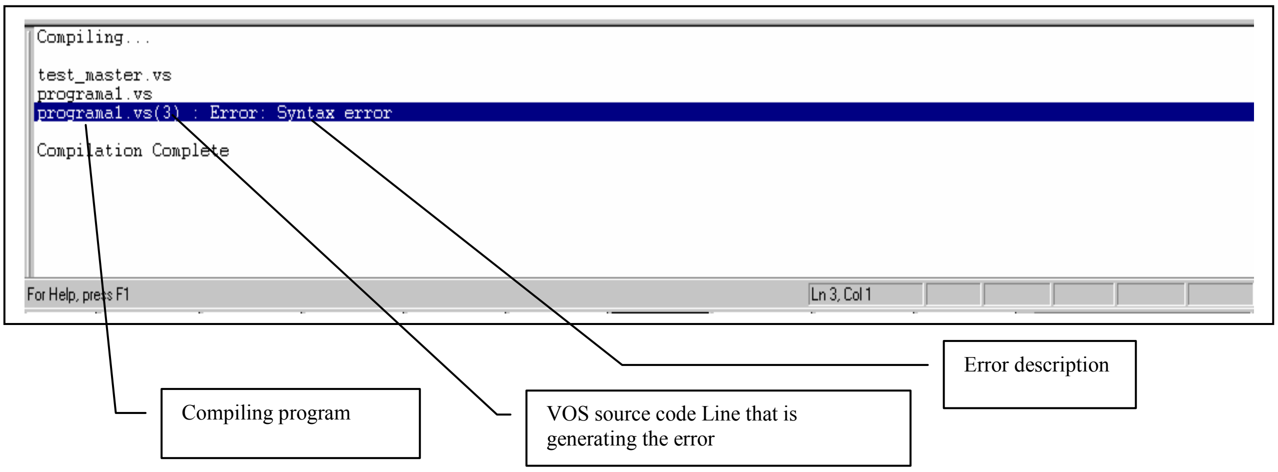
Default Program to run if DNIS phone number is unassigned: CallNotAnswered

For the rest of the DNIS or ANIS (that we don´t have listed) we can associate a program or just to refuse the call.

## 6.25. Graphical VOS. Executing the application

The menu option “**Run**” includes all the options to required to compile and execute the VOS program. The compiler process is the task where VOS checks and translates from VOS code to **p-code**. VOS will compile each application in the project. The compilation application is called: **VLC8.exe**[1]

During the compilation process, there is a state window that displays status, errors and warnings generated during compilation and linking:



```
Compiling...

test_master.vs
programa1.vs
programa1.vs(3) : Error: Syntax error

Compilation Complete
```

Once the program is compiled (without errors) the application is ready to execute.

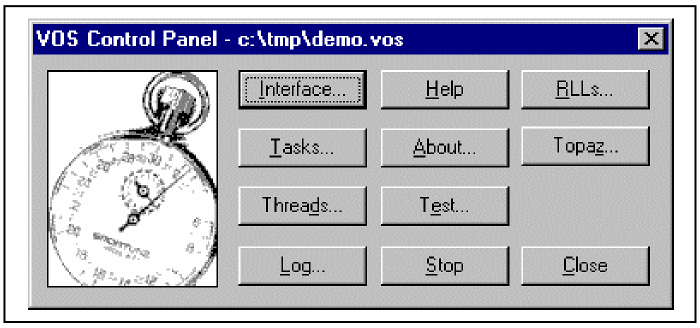
During Execution, VOS will launch the VOS.exe application that it will interpret the compiled VOS code. To do that we could use the **F5** key or the arrow

[1] VLC8.exe and VOS8.exe can be configured in the menu Project->Options.

The VOS Runtime has several configurations:

### *6.25.1. User Interface .- Configuration*

When VOS starts, the default user interface is a dialog called the VOS Control Panel. The VOS Control Panel looks like this:



**There is a reference to the current file .VOS in the task bar.**

The configuration for the compiler (vlc) and the runtime (vos) is done in two files: VOS.ini and the files with the extension .VOS.

#### *6.25.1.1. Configuration file .VOS*

Starting with version 7, VOS and VLC settings are consolidated into two files: the VOS.INI and a new file type called a VOS Settings file and given the .vos extension.

A Settings file is very similar to an INI file, except that while the VOS.INI file applies to every time instance of the VOS runtime engine, you can have a different Settings file for each of your VOS projects. When you run a VOS program, you can specify which settings file to use. If you haven’t explicitly specified a settings file, VOS uses the default Settings file specified in the [Settings] section of VOS.INI. Moving the Settings file out of the Windows directory and allowing an arbitrary filename makes it easier to work on several different systems on a single PC.

The VOS Settings file format is based on the Windows INI format, and internally VOS and VLC use the Windows API functions for manipulating INI files, so the same syntax for specifying section names, key names, values and comments all apply.

This file can be edit from three places:

- From the Graphical VOS, Menu -> Project->**VOS Settings**.
- From the **VOS Control Panel**, right button click.
- Using **NotePad** and editing **the .VOS ini** file directly.

Sections in the Settings file include

**[AutoStart]**

The [AutoStart] section of the VOS settings file specifies up to 16 .vx files to be loaded when VOS is started using this settings file. For example:

```
[AutoStart]
VXProgram1=C:\VoiceMail\VX\Master.vx
VXProgram2=Q:\Shared Vos Files\VX\Database.vx
VXProgram3=Q:\Shared Vos Files\VX\Pager.vx
```

**[Buf]**

The [Buf] section of the VOS settings file lets you set the number of buffers for the buf_use family of functions.

**buf_use. Sample.**

Count = 10

**[Colors]**

The [Colors] section of the VOS settings file stores the 16 custom VOS Box colors you can set with the VOS Control Panel.

| **Entry** | **Meaning** |
|---|---|
| Custom0 | Custom color 0. |
| Custom1 | Custom color 1. |
| Custom2 | Custom color 2. |
| Custom3 | Custom color 3. |
| Custom4 | Custom color 4. |
| Custom5 | Custom color 5. |
| Custom6 | Custom color 6. |
| Custom7 | Custom color 7. |
| Custom8 | Custom color 8. |
| Custom9 | Custom color 9. |
| Custom10 | Custom color 10. |
| Custom11 | Custom color 11. |
| Custom12 | Custom color 12. |
| Custom13 | Custom color 13. |
| Custom14 | Custom color 14. |
| Custom15 | Custom color 15. |

The format of the custom color specification is the hex value of a Win32 RGB value.

**[Consts]**

The [Consts] section of the VOS settings file specifies constants that are declared when VLC is started.

```
[Consts]
Const1=<Name>[:Value]
```

```
Const2=<Name>[:Value]
... etc. for Const3, Const4, ...
```

**[DateTime]**

The [DateTime] section of the VOS settings file lets you override VOS's daylight savings time settings.

| **Entry** | **Meaning** |
|---|---|
| DST | Set to 0 to indicate that standard time is in effect, or to a value greater than 0 to indicate that daylight savings time is in effect, or to a value less than zero to have VOS compute whether standard time or daylight savings time is in effect for the given date. |

**[DBF]**

The [DBF] section of the VOS settings file lets you set VOS parameters for working with DBF databases.

| **Entrada** | **Significado** |
|---|---|
| AllowUnsafe [2] | Unsafe db_fput calls allowed? 1=Yes 0=No |
| MaxBases | Maximum number of active bases. |
| MaxBaseHandles | Maximum number of base handles (formerly known as descriptors). |
| MaxFields | Maximum number of fields in one database. |
| MaxRecHandles | Maximum number of record handles (formerly known as descriptors). |
| MaxRecBytes | Record length. db_reclen must be larger than the database record length. Use dmpdbf.exe to examine the database record length. |
| MaxRecs | Maximum number of active records. |

[2] AllowUnsafe must be set to 1 to allow sharing. The main use for this parameter is to allow backwards-compatibility with applications which were designed to share handles across tasks.

**[Exceptions]**

The [Exceptions] section of the VOS settings file controls exception handling.

| **Entry** | **Meaning** |
|---|---|
| Handler | The possible values of Handler are 1 and 2, where 1 means that VOS will not do any exception handling or logging, and 2 means that VOS will handle the exception (log the stack trace of the exception in EXCEPT.RPT file) and exit immediately with the ERRORLEVEL 3, which can be used in batch files.<br><br>The default VOS behavior in case of an exception is to log the stack trace of the exception in EXCEPT.RPT file and pass the control to the next exception handler in line, which in most cases will be Dr.Watson or Visual C++ debugger. |

**[File]**

The [File] section of the VOS settings file lets you set file and record locking parameters for DBF database files.

| **Entry** | **Meaning** |
|---|---|
| MaxLocks | Maximum number of file locks. |

**[FunFileDirs]**

The [FunFileDirs] section of the VOS settings file lets you specify up to eight directories in which VLC should search for function files.

| **Entry** | **Meaning** |
|---|---|
| Dir1 | First function file path name. |
| Dir2 | Second function file path name. |

...

| | |
|---|---|
| Dir8 | Eighth function file path name. |

VLC checks the directories from Dir1 to Dir8--for example, function files in the Dir1 directory would have precedence over files in the Dir2 directory. This is important to understand particularly if you have functions of the same name in two of the specified directories. Suppose you have two files named myfunc.fun, one in the C:\functions directory, and the other in D:\functions. If the Settings file for your project looks like this

```
[FunFileDirs]
Dir1=D:\functions
Dir2=C:\functions
```

when your program compiles, VOS will use the function stored in the D:\functions directory

**[Glb]**

The [Glb] section of the VOS settings file lets you set parameters for numbered global variables.

The following entries don't affect named global variables.

| **Entry** | **Meaning** |
|---|---|
| Count | The maximum number of numbered global variable to allow. |
| VarBytes | The maximum size of each numbered global variable, in bytes |

**[GUI]**

The [GUI] section of the VOS settings file lets you control the appearance of the VOS user interface. You can set these entries through the VOS Control Panel User Interface dialog.

| Entry | Meaning |
|---|---|
| AboutBoxText | String that appears in the VOS About Box. |
| CtlPanelTitle | Sets the title for the Control Panel. |
| ConfirmStop | Should a dialog box confirm before stopping VOS? Set to 1 to see the confirmation. Set to 0 to stop without a confirmation dialog. |
| EnableDialogs | 1 enables, 2 disables dialogs. By default, VOS ensables all dialogs (message boxes). Disabling dialogs allows VOS to exit immediately, without waiting for the operator to click OK. |
| ShowCtlPanel | Should VOS display the Control Panel when running? Set to 1 to show the Control Panel, or 0 to hide it. |
| StopIfNoTasks | Should VOS stop if there are no tasks running? Set to 1 to stop or 0 to continue running if there are no tasks running. |

**[IncludeFileDirs]**

The [IncludeFileDirs] section of the VOS settings file lets you specify up to eight directories in which VLC should search for include files.

| Entry | Meaning |
|---|---|
| Dir1 | First include file path name. |
| Dir2 | Second include file path name. |
| ... | |

| | |
|---|---|
| Dir8 | Eighth include file path name. |

VLC checks the directories from Dir1 to Dir8--for example, include files in the Dir1 directory would have precedence over files in the Dir2 directory. This is important to understand if you have files of the same name in two of the specified directories. Suppose you have two files named project.inc, one in the C:\include directory, and the other in D:\include. If the Settings file for your project looks like this

```
[IncludeFileDirs]
Dir1=D:\include
Dir2=C:\include
```

when your program compiles, VOS will use the file in the D:\include directory

**[Load]**

The [Load] section of the VOS settings file lets you control which DLL VOS loads by default.

The name and path of the Debug and Non-debug DLLs are set in the [DLLs] section of the VOS.INI file.

| **Entry** | **Meaning** |
|---|---|
| DebugDLL | Should VOS load the Debug DLL (that is, start in Debug mode) by default? Set to 1 to start in Debug mode or to 0 to start in Non-debug mode by default. |

**[Log]**

The [Log] section of the VOS settings file lets you set parameters for the VOS log file.

| Entry | Meaning |
|---|---|
| Append | Should new log file entries be appended to an existing log? Set to 1 to append new entries or to 0 to overwrite any old log when you start VOS. |
| Buffer | 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 bunch of lines at one time. If not buffered, VOS writes each line as it comes. |
| Commit | Boolean. Set to 1 to commit the log file to disk each time a line is written. (Committing is the equivalent of asking Windows not to buffer.) Commit=1 is required only if Windows doesn't complete a write in the case of a crashed process. This is usually not necessary because Windows will write any buffered output to the file even if a process crashes. Since this will be much slower, set to 1 only if really needed. |
| Dir | Directory in which the vos?.log files are stored. |
| MaxSizeKb | Sets the maximum size of the VOS log. |

**[Msg]**

The [Msg] section of the VOS settings file lets you set message function parameters.

| Entry | Meaning |
|---|---|
| MaxCount | Maximum number of messages. |
| MaxChars | Max characters in a message+1 for terminating nul byte |
| Msg_IntervalMs | |

.

**[NTX]**

The [NTX] section of the VOS settings file lets you set DBF database index parameters.

| **Entry** | **Meaning** |
|---|---|
| Buffers | Cache buffers per index: See Settings File Index Requirements. |
| Clipper | .NTX: 0=older VOS 1=free list 2=key sort 3=both |
| MaxKey | Index field length: See Settings File Index Requirements. |
| MaxOpen | Number of index files: See Settings File Index Requirements. |
| PageErrFatal | Is a page error fatal? Set to 1 to make a page error fatal or 0 to make page errors not fatal. |

**[Prio]**

The [Prio] section of the VOS Settings file lets you set Windows priorities for the VOS process, primary thread, and GUI thread. Windows priorities are explained in detail in the MSDN Library--for more information, see the SetThreadPriority and SetThreadPriorityBoost Win32 API functions in the MSDN Library.

The SetThreadPriority and SetThreadPriorityBoost Win32 API functions are called when VOS starts (using the values set in this section of the VOS Settings file) and again when a setting is changed via the Control Panel Set Priorities dialog--you don't have to restart VOS to change priorities.

| Entry | Meaning |
|---|---|
| GuiBoost | Enables or disables priority boosting for the VOS GUI thread. Set to 1 to enable boosting or 0 to disable it.<br><br>See SetThreadPriorityBoost (a Win 32 API function) in the MSDN Library for an explanation of priority boosting. |
| GuiThread | Sets the priority value for the VOS GUI thread.<br><br>See SetThreadPriority (a Win 32 API function) in the MSDN Library for information on thread priorities and a list of valid priority values. |
| PrimaryBoost | Enables or disables priority boosting for the VOS primary thread. Set to 1 to enable boosting or 0 to disable it.<br><br>See SetThreadPriorityBoost (a Win 32 API function) in the MSDN Library for an explanation of priority boosting. |
| PrimaryThread | Sets the priority value for the VOS primary thread.<br><br>See SetThreadPriority (a Win 32 API function) in the MSDN Library for information on thread priorities and a list of valid priority values. |
| ProcessClass | Sets the priority value for the VOS process class.<br><br>See SetThreadPriority (a Win 32 API function) in the MSDN Library for information on thread priorities and a list of valid priority values. |

**[R4]**

The [R4] section of the VOS settings file lets you set parameters for legacy VOS functions such as sc_, DTI_ etc.

If your application does not use these legacy functions, omit this section from your settings file.

| Entry | Meaning |
|---|---|
| Enable | Should VOS load the R4 drivers? Set to 1 to load the R4 drivers or 0 to not load them. |
| fxSendFiles | Maximum number of fax files in a single FaxSend. |
| gcIEBytes | Maximum number of bytes that will be used in an ISDN information element. |
| scEnablePCPA | Should the system use Perfect CPA? Set to1 to use Perfect CPA, or to 0 otherwise. |
| scNshgup | Should a continuous tone be treated as hang-up? Set to 1 to treat a continuous tone as a hang-up, or to 0 otherwise. |
| scPhraseWords | Maximum words in a phrase. Must be > 0 to use phrases |

## [RLLs]

The [RLLs] section of the VOS settings file lets you specify which RLLs to load when running VOS or VLC.

| Entry | Meaning |
|---|---|
| RLL1 | First RLL path name. |
| RLL2 | Second RLL path name. |
| ... | |
| RLL8 | Eighth RLL path name. |

## [SearchPaths]

The [SearchPaths] section of the VOS settings file lets you set paths to be searched for IPFs, Par files, and VX files.

If you want to list more than one path for one of the SearchPaths entries, separate each path with a semicolon:

```
[SearchPaths]
IPF=.;.\IPF;c:\Projects\IPFs;
```

| **Entry** | **Meaning** |
|---|---|
| IPF | Sets the path for locating Indexed Prompt Files. |
| Par | Sets the path for locating PAR files. |
| VX | Sets the path for locating .vx files in the VOS command line or started via chain, spawn or exec. |

**[Sem]**

The [Sem] section of the VOS settings file lets you set Semaphore parameters.

| **Entry** | **Meaning** |
|---|---|
| Count | Maximum number of numbered semaphores. |

**[Ser]**

The [Ser] section of the VOS settings file lets you set serial port parameters.

| **Entry** | **Meaning** |
|---|---|
| BuffKb | COM port buffer size, in bytes. |
| Ports | Number of COM ports to support. |

| | |
|---|---|
| Ser_IntervalMs | The interval, in milliseconds, for polling serial ports (default = 50). |

## [SRL]

The [SRL] section of the VOS settings file lets you set parameters for Standard Runtime Library (SRL) events.

| **Entry** | **Meaning** |
|---|---|
| Srl_IntervalMs | The interval in milliseconds for polling Dialogic SRL events (default=25). |

## [Task]

The [Task] section of the VOS settings file lets you set task management parameters.

| **Entry** | **Meaning** |
|---|---|
| MaxCount | Maximum number of tasks. |
| MaxName | Maximum length of VOS task names. |

## [Trace]

The [Trace] section of the VOS settings file lets you set tracing options.

| **Entry** | **Meaning** |
|---|---|
| ActiveX | Should VOS provide detailed tracing of object accesses (available in Debug mode only)? Set to 1 for this tracing, or set to 0 to disable it. |
| Builtins | Should VOS Trace all built-in functions? Set to 1 to trace or |

| | |
|---|---|
| | 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 will not show in the log. |
| Layer | Log DOS-to-Windows translation layer for "legacy" functions sc_, DTI_ etc. |
| OutputToVosBox | Should tracing information be displayed in the VOS Box as well as written to the log file? Set to 1 to display the tracing information or to 0 to just write it to the log file. |
| Override | Should the entries in this Settings file override tracing options set with the trace() function in individual programs? Set to 1 to override or to 0 to use the program's settings. |
| Pcode | Should VOS trace all p-code instructions? Set to 1 to trace or to 0 to disable p-code tracing. (P-code tracing will create a large amount of logging.) |
| RLLs | Should VOS trace all RLL calls? Set to 1 to trace or to 0 to disable tracing of RLLs. |
| Stack | Should VOS include the stack in p-code tracing? Set to 1 to include the stack or to 0 to omit it. |
| Structs | Should VOS trace API structure members? Set to 1 to trace or to 0 to omit structures. |
| Vars | Should VOS include variables in p-code tracing? Set to 1 to include variables or to 0 to omit them. |

**[TrayIcon]**

The [TrayIcon] section of the VOS settings file lets you control the system tray icon that appears when VOS is running. You can set these entries through the VOS Control Panel User Interface dialog.

| **Entry** | **Meaning** |
|---|---|
| Animate | Should the tray icon be animated? Set to 0 to prevent animation. Set to 1 to allow animation. |
| HWND | Used internally by VOS. |
| Show | Should the system tray icon show while VOS is running? Set to 0 to hide the icon. Set to 1 to show the icon. |
| Tooltip | Sets the text that appears in the tooltip when a user's mouse is over the system tray icon. |

**[VLC]**

The [VLC] section of the VOS settings file lets you set VOS Language Compiler options.

| **Entry** | **Meaning** |
|---|---|
| GenerateDebugSymbols | Should VLC generate debug symbols? Set to 1 to enable or 0 to disable. By default, debug symbols are enabled. |
| GenerateLineMarks | Should VLC generate line marks? Set to 1 to enable or 0 to disable. By default, line marks are generated. |
| ListFileName | Specifies the name of the list file to create. If this entry is not specified, no list file is created. |

| | |
|---|---|
| ReportUnusedVariables | Set to 1 to enable or 0 to disable. By default, unused variables are not reported. |
| LongFileNameSupport | Should VLC use long file names when looking for function files?<br><br>If set to 1, VLC uses the full function name as a long file name. If not found, report an error. For example, if a function is called MyFunction() VLC looks for a file called MyFunction.fun<br><br>If set to 2, VLC uses the full function name as a long file name. If the name is not found, truncate the function name to 8 characters and look for that .fun file. If not found, report an error. For example, if a function is called MyFunction() VLC looks for a file called MyFunction.fun. If VLC doesn't find MyFunction.fun, it looks for MyFuncti.Fun.<br><br>If set to 3, VLC always truncates the function name. For our exaple function, MyFunction(), VLC only looks for MyFuncti.fun.<br><br>By default, long file names are enabled. |
| StackDepth | Set stack depth. |
| StackSize | Set the stack size, in bytes. The default is 2048. |
| Verbose | Display progress of compile, a value from 0 to 9, to show increasing detail. |

**[VLs]**

The [VLs] section of the VOS settings file lets you specify which VL files to load when running VOS or VLC. A Vlc Library, file extension .VL, is a collection of function definition (.FUN) files in one file.

```
[VLs]
Lib1=<VL path name>
... etc for Lib2, Lib3...
```

**[VosBox]**

The "VOS Box" window is a 25 x 80 character display that emulates the appearance of an MS-DOS PC. It is provided for backwards compatibility with older applications written for VOS for DOS.

The [VosBox] section of the VOS settings file lets you set options for the VOS Box. You can also set these entries through the VOS Control Panel User Interface dialog.

| **Entry** | **Meaning** |
|---|---|
| BkBlue | Intensity of blue in background color (0 to 255). |
| BkGreen | Intensity of green in background color (0 to 255). |
| BkRed | Intensity of red in background color (0 to 255). |
| EnableCtrlBreak | Should typing Ctrl+Break stop VOS? Set to 1 to enable or 0 to disable. By default, Ctrl+Break is disabled. |
| Font | Used to choose a font for screen display; this is necessary for displaying international characters. |
| FontBlue | Intensity of blue in text font color (0 to 255). |

| | |
|---|---|
| FontGreen | Intensity of green in text font color (0 to 255). |
| FontItalic | Should the font be italic? Set to 1 for italic, or 0 for regular. |
| FontRed | Intensity of red in text font color (0 to 255). |
| FontSize | Sets the font size, in points, for screen display. |
| FontWeight | Font weight (see Win32 CreateFont function for valid values). |
| Show | Should the VOS Box show when VOS is running? Set to 1 to enable or 0 to disable. |
| ShowTime | Should the VOS Box show the current time? Set to 1 to enable or 0 to disable. |
| Title | This string is used as the title for the VOS Box. If you don't set a Title, VOS uses "VOS Box" by default. |

##### *6.25.1.2. File* ***VOS.INI***

The file VOS.ini stores configurations that affect VOS. These configurations are as follow:

**[Settings]**

The Default entry in this section of the VOS.INI file points to the default Settings file:

```
[Settings]
Default=c:\Vos\Settings\Default.vos
```

**[DLLs]**

Entries in this section of the VOS.INI file point to the location of the Debug and Non-Debug VOS DLLs.

```
[DLLs]
Debug=c:\Vos\Bin\Vosd.dll
NonDebug=c:\Vos\Bin\Vos.dll
```

**[Service]**

Entries in this section of the VOS.INI file specify service dependencies and other settings needed when running VOS as a Windows NT/2000 service.

```
[Service]
GroupName=VoiceMail
Dependencies=Dialogic;Telephony
GroupOrderTag=2
```

# 7. CT ADE Architecture (Topaz) , Introduction

The first question is: **What is CT ADE Architecture (TOPAZ) ?**

Before I answer this question I would like to review the problems posed by the CTI applications.

When developing telephony applications, we need to keep in mind: Type of telephone lines, type of boards and the differents API's involved. There are 3 kinds of interfaces:

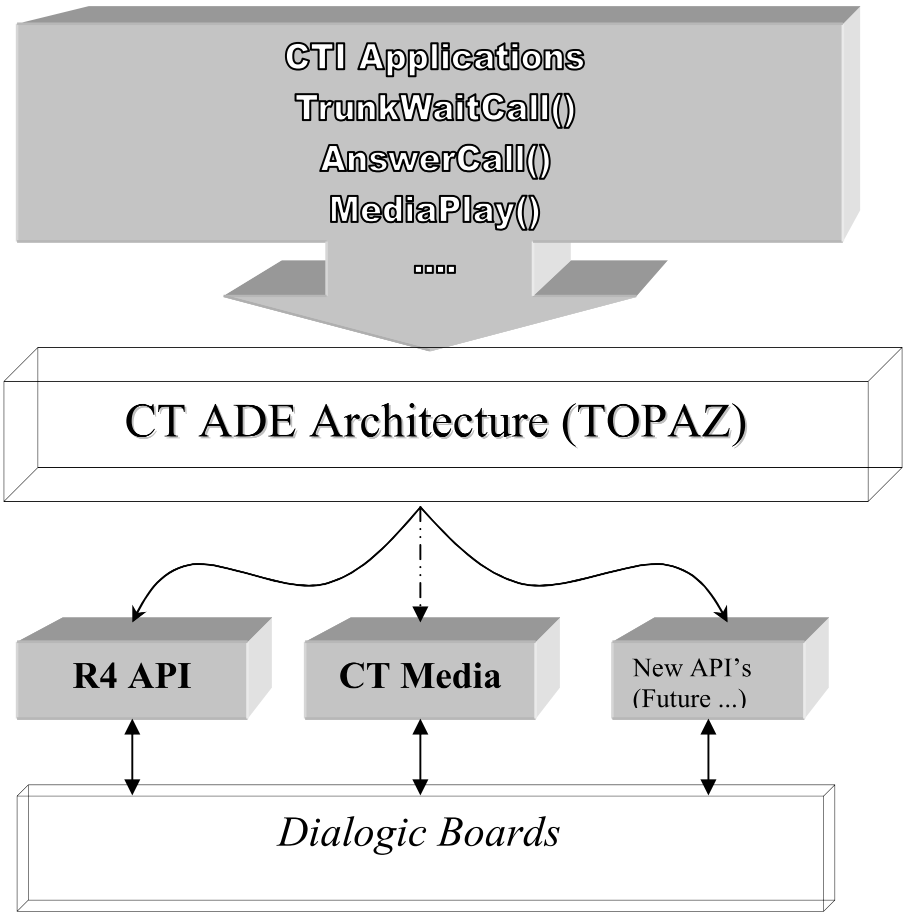
- Analog
- Digitals (R2, ISDN, SS7, etc…)
- IP (Voice over IP).

For example, if we need to develop and maintain the same application for each type of interface, we would have to build 3 diferent applications: Analog version, Digital version, and IP version. In addition, each of these applications depend directly on the board API. Furthermore, if we want to deploy the application on CT Media (for example) the we would have to once again rewrite the entire application from scratch.

Here is where CTADE_A provides a great advantage. With CTADE_A we have a group of easy commands, that **allow us to develop one application for any type of trunk and any kind of API**. This concept is called **API transparency**.

## How can CTADE_A achieve API transparency?

The schematic shows where TOPAZ is in the system and how it “isolates” the application from the API.



As we see in the above diagram, TOPAZ is between the API’s and the application. TOPAZ **detecta the API and installed boards (using the Topaz Profile - we will see later).**

TOPAZ is **RESOURCE** oriented, it deals directly with the telephony resources in the system .

Well, this is what we call a system resource oriented . **What is a resource ?**

A resource is something that refers to an element or a group of elements where each of one represents an entity of elements to be shared and use. The first thing that we have to define is the type of resources. In VOS we have the following resources:

- Trunk Resource: Line Interface resource.
- Media Resource:  Play, Record, Tones generation/detection resources.
- FAX Resource: Resources to send and receive FAXES
- Voice recognition Resources: Speech recognition resources.
- Text-to-Speech Resources: Text to speach resources.
- Conferencing Resources: Conferencing to arrange and control resources.

As we can see in the follow VOS Sample code, the functions that refer to the resources always start with the name of the resource:

| | |
|---|---|
| **MEDIA FUNCTIONS** | MediaPlayFile()<br>MediaGetDigitBuffer()<br>MediaRecordFile()<br>MediaWait()<br>.... |
| **TRUNK FUNCTIONS** | TrunkWaitCall()<br>TrunkAnswerCall()<br>TrunkMakeCall()<br>TrunkWait()<br>... |
| **FAX FUNCTIONS** | FaxAbort()<br>FaxAddTextFile()<br>FaxWait()<br>... |

## 7.1. Graphical VOS. CT ADE Architecture Unleashed.

Now, we know how to create a project and how to add the necessary files to the proyect. In addition, we know how to configure a VOS project.

Next, we will try to understand CTADE_A and the VOS language. As we already know, CTADE_A is a collection of software modules that implement a very light layer between the CTADE_A commands (MediaPlayFile, MediaRecord, and so on…) and the installed CTI API.

CTADE_A is heavily resource oriented. As we saw in the introduction, there are several types of resources (Trunk, Media, Fax, ASR, TTS, Conference) that use different technologies from the Dialogic boards.

Each of these resources contain several internal states whose transitions belong to the different resource live phases. For further information about these states, please refer to *Graphical VOS User's Guide en la sección CTADE_A->Resource Stats.*

### *7.1.1. CTADE_A. Resources and Resource Index Numbers.*

Many VOS functions control CTADE_A Resources. For example, the MediaPlayFile function plays a sound file on the current Media Resource. Now, we will introduce the Resource concept and will discuss how Resource numbering works.

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.

Each Resource is identified by a Resource index number. Generally, you won't have to worry about Resource index numbers, since VOS reserves and routes Resources as your application needs them, but if you want to control how your applications use Resources, you'll specify them by their index number. Index numbers start at 0 for each Resource type, and are numbered independent of other Resource types: a VOS task could easily be using Trunk Resource 1 and Media Resource 4.

In Addition, there are several functions to obtain information about the Indexes of our resources:

- ***ResourceType*****GetIndex,** Returns the index number of the current resource, e.g. MediaGetIndex, TrunkGetIndex
- ***ResourceTypeGetCount,*** Returns the total number of resources, e.g. MediaGetCount, TrunkGetCount.

## *7.1.2. CTADE_A Functions overview.*

In CTADE_A, all the functions come in groups of resources. Each type of CTADE_A resources form groups of functions that refers to specific features. Normally, all resources in the system have a number and we have a index to address each resource. There are 2 ways to ask for a resource, automatic and manual. Example:

| **Automatic request of resources** | **Manual request of resources** |
|---|---|
| **TrunkWaitCall**(); | **TrunkUse**(0); |
| **TrunkAnswerCall**(); | **TrunkWaitCall**() |
| if (**TrunkGetState**() strneq "Connected") | **TrunkAnswerCall**(); |
| voslog("Unexpected trunk state ",**TrunkGetState**()); | |
| stop; | if (**TrunkGetState**() strneq "Connected") |
| endif | voslog("Unexpected trunk state |
| | ",**TrunkGetState**()); |
| | stop; |
| **MediaPlayFile**("HelloInbound.vox"); | endif |
| | **MediaUse**(0); |
| **TrunkDisconnect**(); | **MediaPlayFile**("HelloInbound.vox"); |
| restart; | |
| | **TrunkDisconnect**(); |
| | **MediaUnUse**(0); |
| | **TrunkUnUse**(0); |
| | Restart; |

Here we reserve automatically a Trunk resource

When we call here the function Media, if there is no current Media resource, TOPAZ will assign a resource to us.

Here in an explicit way, we are asking for the Trunk resource number 0. If another task has this resource, we will get a error.

Here in the same way that with the Trunk, we ask for the Media resource 0.

*7.1.2.1. CTADE_A Functions. Asynchronous Mode.*

By default, VOS always wait until all CTADE_A functions finish his execution (synchronous way). Example:

**MediaPlayFile**("LeaveMsg.vox");

**MediaRecordFile**("Message");

By default, MediaPlayFile will return the execution when the vox files is finish or another termination condition (DTMF, silence, etc…).

However, in many occasions could be attractive to call the function and continue the execution of the program (asynchronous execution). Example:

**MediaPlayFile("Welcome.vox");**

**Balance=CheckBalance(AccountNo);**

**MediaPlayFile(Balance&".vox");**

Imagine that the function **CheckBalance** take between 1 and 10 secs to execute (depending on how busy the DB server is). In the worst case, the person that is calling into the system will have to wait until 10 secs between the first play and the result of the balance.

Using the asynchronous execution method we can avoid this problem using the function **MediaEnableAsync()** as follow:

```
MediaEnableAsync();
MediaPlayFile("Welcome.vox");
Balance=CheckBalance(AccountNo);
MediaWait();
MediaPlayFile(Balance&".vox");
....
```

This function waits until the state of the Media Resource comes to **Idle**

*7.1.2.2. CTADE_A Functions. Trunk Resource & Functions*

The Trunk functions are used to manage our Trunk resources in the system in order to do things as follow:

- Make outgoing calls.
- Make Call Progress Analisys
- Answer or reject incomming calls.
- Obtain call information (ANI, DNIS, Caller Name, etc...)
- Hang up the call

As you can see, the Trunk resources are responsible for all Call Control. A Trunk Resource processes a single stream of audio, so each of the following is considered a single Trunk Resource:

- The trunk interface for a single analog line.
- One E1/T1 time-slot
- An MSI Station
- SimPhone's simulated trunk line (always Index number 0).

The functions to control the Trunk Resources:

| | | |
|---|---|---|
| TrunkAbort | TrunkAbort([ResIndex]) | Terminates any function in progress on a Trunk Resource. |
| TrunkAnswerCall | TrunkAnswerCall() | Answers an incoming call on the current Trunk Resource. |
| TrunkBlock | TrunkBlock() | Prevents calls from coming in on the current Trunk Resource. |
| TrunkClearAllDeferOnHangup | TrunkClearAllDeferOnHangup() | Calls onhangup if one or more deferred hang-ups did occur. |
| TrunkClearDeferOnHangup | TrunkClearDeferOnHangup() | Calls onhangup if deferred hang-up did occur. |
| TrunkDeferOnHangup | TrunkDeferOnHangup() | Prevents VOS from jumping to onhangup when a caller hang-up is detected. |
| TrunkDisableAsync | TrunkDisableAsync() | Disables asynchronous mode. |
| TrunkDisableCallAnalysis | TrunkDisableCallAnalysis() | Disables call progress analysis on the current Trunk Resource. |
| TrunkDisableIncomingCalls | `TrunkDisableIncomingCalls()` | Prevents incoming calls from arriving on the current Trunk Resource. |
| TrunkDisconnect | TrunkDisconnect() | Disconnects a call on the current Trunk Resource. |

| | | |
|---|---|---|
| TrunkEnableAsync | TrunkEnableAsync([ResIndex]) | Enables asynchronous mode. |
| TrunkEnableCallAnalysis | TrunkEnableCallAnalysis() | Enables call progress analysis on the current Trunk Resource. |
| TrunkEnableIncomingCalls | `TrunkEnableIncomingCalls()` | Allows incoming calls on the current Trunk Resource. |
| TrunkGetANI | ANI = TrunkGetANI() | Returns ANI digits for an incoming call on the current Trunk Resource. |
| TrunkGetBin | BinValue = TrunkGetBin(ParmID, BufNr, BytesToRead) | Returns the binary value of the specified CTADE_A Trunk parameter. |
| TrunkGetBool | BoolValue = TrunkGetBool(ParmID) | Returns the Boolean value of the specified Topaz Trunk parameter. |
| TrunkGetCallerName | Name = TrunkGetCallerName() | Returns the caller name for an incoming call on the current Trunk Resource. |
| TrunkGetCount | Count = TrunkGetCount([TopazTasks]) | Returns the number of Trunk Resources in Topaz, assigned to the current task, or assigned to another task. |
| TrunkGetDNIS | DNIS = TrunkGetDNIS() | Returns DNIS digits for an incoming call on the current Trunk Resource. |
| TrunkGetIndex | ResIndex = TrunkGetIndex() | Returns the Resource index number for the current Trunk Resource. |
| TrunkGetInt | IntValue = TrunkGetInt(ParmID [, Arg1, Arg2, ... ArgN]) | Returns the integer value of the specified Topaz Trunk parameter. |

| | | |
|---|---|---|
| TrunkGetOwnerTask | TaskID = TrunkGetOwnerTask(ResIndex) | Returns the task ID of the task in which the specified Trunk Resource is open. |
| TrunkGetState | State = TrunkGetState([Type]) | Returns the current state of the current Trunk Resource. |
| TrunkGetString | StrValue = TrunkGetString(ParmID) | Returns the string value of the specified Topaz Trunk parameter. |
| TrunkGetTech | TechType = TrunkGetTech([Type]) | Returns information on the technology used by the current Trunk Resource. |
| TrunkGetTechIndex | TechIndex = TrunkGetTechIndex() | Returns the Technology Index for the current Trunk Resource. |
| TrunkIsANIAvailable | ANIStatus = TrunkIsANIAvailable() | Returns a description of the ANI digits available for an incoming call on the current Trunk Resource. |
| TrunkIsANISupported | Support = TrunkIsANISupported() | Checks ANI support for the current Trunk Resource. |
| TrunkIsCallerNameAvailable | Available = TrunkIsCallerNameAvailable() | Checks whether a caller name is available for an incoming call on the current Trunk Resource. |
| TrunkIsCallerNameSupported | Support = TrunkIsCallerNameSupported() | Checks whether caller names are supported by the current Trunk Resource. |
| TrunkIsDNISAvailable | Available = TrunkIsDNISAvailable() | Determines if DNIS is available for an incoming call on the current Trunk Resource. |
| TrunkIsDNISSupported | Support = TrunkIsDNISSupported() | Checks DNIS support for the current Trunk Resource. |

| | | |
|---|---|---|
| TrunkListenTo | TrunkListenTo(ResType, ResIndex) | Routes the current Trunk Resource to another Resource. |
| TrunkMakeCall | TrunkMakeCall(PhoneNumber) | Places an outbound call on the current Trunk Resource. |
| TrunkRejectCall | TrunkRejectCall() | Rejects an incoming call on the current Trunk Resource. |
| TrunkReset | TrunkReset([ResIndex]) | Resets a Trunk Resource. |
| TrunkS100HandOff | TrunkS100HandOff(ASI [, WaitForReturn, CatchTag]) | CT Media only: hands the call to a specified application. |
| TrunkS100Retrieve | TrunkS100Retrieve([Cause]) | CT Media only: retrieves a call group that was previously handed off. |
| TrunkS100Return | TrunkS100Return([CatchTag]) | CT Media only: returns a call group to a previous owner. |
| TrunkS100WaitForReturn | TrunkS100WaitForReturn() | CT Media only: waits indefinitely for a handed-off call group to return. |
| TrunkSetBin | TrunkSetBin(ParmID, BufNr, BytesToWrite) | Sets the specified Topaz Trunk parameter to a binary value. |
| TrunkSetBool | TrunkSetBool(ParmID, Value) | Sets the specified Topaz Trunk parameter to a Boolean value. |
| TrunkSetInt | TrunkSetInt(ParmID, [Arg1, Arg2, ... ArgN,] Value) | Sets the specified Topaz Trunk parameter to an integer value. |
| TrunkSetString | TrunkSetString(ParmID, Value) | Sets the specified Trunk parameter to a string value. |

| | | |
|---|---|---|
| TrunkUnblock | TrunkUnblock() | Allows incoming calls on the current Trunk Resource. |
| TrunkUnlisten | TrunkUnlisten() | Disconnects a listen connection |
| TrunkUnUse | TrunkUnUse() | Frees the current Trunk Resource for so that it can be used by other tasks. |
| TrunkUse | `TrunkUse([ResIndex])` | Reserves a Trunk Resource for the task. |
| TrunkUseNext | `TrunkUseNext()` | Reserves the next available Trunk Resource for the task. |
| TrunkUseNoWait | TrunkUseNoWait([ResIndex]) | Reserves a Trunk Resource for the task and returns immediately if it is not available. |
| TrunkWait | TrunkWait([ResIndex]) | In asynchronous mode, waits for a function to complete. |
| TrunkWaitCall | TrunkWaitCall() | Waits for an incoming call on the current Trunk Resource. |
| TrunkWaitDisconnect | TrunkWaitDisconnect([Timeout]) | Waits for the caller to hang up or otherwise be disconnected from the current Trunk Resource. |

*7.1.2.3. CTADE_A Functions. Media Resources & Functions*

Media Resources control playing and recording of sound files and tones and getting DTMF digits from callers. The Media Resources normally are used to:

- Playing Sound Files and Playing an Indexed Prompt describe playing audio files to a caller Reproducir frases numerícas (funciones spk_)
- Speaking Phrases introduces you to phrases that contain variable information: *"Your balance is three hundred dollars and seventeen cents."*
- Recording Sound Files shows you how to record audio from a caller to an audio file.
- Using Stop Tones describes setting touch tone digits that, when dialed by the caller, will interrupt playing or recording.
- Getting Digits describes retrieving touch-tones from a caller.
- Playing Tones shows how to play general tones to a caller.

The functions to control Media Resources are:

| | | |
|---|---|---|
| MediaAbort | MediaAbort([ResIndex]) | Terminates any function in progress on a Media Resource. |
| MediaClearDigitBuffer | MediaClearDigitBuffer() | Discards any digits in the current Media Resource's channel buffer. |
| MediaDisableAsync | MediaDisableAsync([ResIndex]) | Disables asynchronous mode. |
| MediaEnableAsync | MediaEnableAsync([ResIndex]) | Enables asynchronous mode. |
| MediaGetBool | BoolValue = MediaGetBool(ParmID) | Returns the Boolean value of the specified Topaz media parameter. |
| MediaGetCount | Count = MediaGetCount([TopazTasks]) | Returns the number of Media Resources in Topaz, assigned to the current task, or assigned to another task. |

| | | |
|---|---|---|
| MediaGetDigitBuffer | Digits = MediaGetDigitBuffer() | Returns the current contents of the digit buffer. |
| MediaGetDigitCount | DigitCount = MediaGetDigitCount() | Returns the number of digits pending in the digit buffer. |
| MediaGetIndex | ResIndex = MediaGetIndex() | Returns the ResourceIndex number for the current Media Resource. |
| MediaGetInt | IntValue = MediaGetInt(ParmID[, Arg1, Arg2, ... ArgN]) | Returns the integer value of the specified Topaz media parameter. |
| MediaGetLanguage | LanguageName = MediaGetLanguage() | Returns the name of the current language. |
| MediaGetOwnerTask | TaskID = MediaGetOwnerTask(ResIndex) | Returns the task ID of the task in which the specified Media Resource is open. |
| MediaGetPlayRecordDuration | Duration = MediaGetPlayRecordDuration() | Returns the length of the last MediaPlayFile or MediaRecordFile. |
| MediaGetState | State = MediaGetState([Type]) | Returns the current state of the current Media Resource. |
| MediaGetString | StrValue = MediaGetString(ParmID) | Returns the string value of the specified Topaz media parameter. |
| MediaGetTech | TechType = MediaGetTech([Type]) | Returns information on the technology used by the current Media Resource. |
| MediaGetTechIndex | TechIndex = MediaGetTechIndex() | Returns the technology index for the current Media Resource. |
| MediaListenTo | MediaListenTo(ResType, ResIndex) | Routes the current Media Resource to another Resource. |
| MediaPlayDate | MediaPlayDate(Value [, Gender]) | Plays a date on the current Media Resource. |
| MediaPlayDualCadence | MediaPlayDualCadenceTone(Freq1, | Plays a dual cadence tone on the current |

| | | |
|---|---|---|
| Tone | Amp1, Freq2, Amp2, OnTime, OffTime, Count) | Media Resource. |
| MediaPlayDualContinuousTone | MediaPlayDualContinuousTone(Freq1, Amp1, Freq2, Amp2, Duration) | Plays a continuous single or dual tone on the current Media Resource. |
| MediaPlayDigits | MediaPlayDigits(Digits) | Generates the specified DTMF digits on the current Media Resource. |
| MediaPlayFile | MediaPlayFile(Filename, [StartPosTenths, Duration, FileSpec]) | Plays a sound file on the current Media Resource. |
| MediaPlayInteger | MediaPlayInteger(Value [, Gender]) | Plays an integer on the current Media Resource. |
| MediaPlayList | MediaPlayList([StartPosTenths, Duration]) | In List mode, plays the queued files. |
| MediaPlayListClear | MediaPlayListClear() | Clears the list of queued files. |
| MediaPlayListModeOff | MediaPlayListModeOff() | Disables List mode. |
| MediaPlayListModeOn | MediaPlayListModeOn() | Enables List mode. |
| MediaPlayMoney | MediaPlayMoney(Value [, Gender]) | Plays a money value on the current Media Resource. |
| MediaPlayOrdinal | MediaPlayOrdinal(Value [, Gender]) | Plays an ordinal value on the current Media Resource. |
| MediaPlayPrompt | MediaPlayPrompt(IPFName, PromptIndex) | Plays a prompt from an Indexed Prompt File on the current Media Resource. |
| MediaPlaySingleCadenceTone | MediaPlaySingleCadenceTone(Freq1, Amp1, OnTime, OffTime, Count) | Plays a single cadence tone on the current Media Resource. |
| MediaPlaySingleContinuousTone | MediaPlaySingleContinuousTone(Freq1, Amp1, Duration) | Plays a continuous single or dual tone on the current Media Resource. |

| | | |
|---|---|---|
| MediaPlayString | MediaPlayString(Value [, Gender]) | Plays a string on the current Media Resource. |
| MediaPlayTime | MediaPlayTime(Value [, Gender]) | Plays a time on the current Media Resource. |
| MediaPlayTone | MediaPlayTone(ToneID) | Plays a tone on the current Media Resource. |
| MediaPlayUserDefined | MediaPlayUserDefined(Type, Value [, Gender]) | Speaks a phrase element of a user-defined type. |
| MediaRecordFile | MediaRecordFile(FileName, [MaxSeconds, MaxSilence, Append, FileSpec]) | Records sound from the current Media Resource to a sound file. |
| MediaReset | MediaReset([ResIndex]) | Resets a Media Resource. |
| MediaSetBool | MediaSetBool(ParmID, Value) | Sets the specified Topaz media parameter to a Boolean value. |
| MediaSetInt | MediaSetInt(ParmID, [Arg1, Arg2, ... ArgN,] Value) | Sets the specified Topaz media parameter to an integer value. |
| MediaSetLanguage | MediaSetLanguage(Language) | Sets the language to be used when speaking phrases. |
| MediaSetString | MediaSetString(ParmID, Value) | Sets the specified media parameter to a string value. |
| MediaUnlisten | MediaUnlisten() | Disconnects a listen connection. |
| MediaUnUse | MediaUnUse() | Frees the current Media Resource so that it can be used by other tasks. |
| MediaUse | `MediaUse([ResIndex])` | Reserves a Media Resource for the task. |
| MediaUseNext | `MediaUseNext()` | Reserves the next available Media Resource for the task. |

| | | |
|---|---|---|
| MediaUseNoWait | MediaUseNoWait([ResIndex]) | Reserves a Media Resource for the task and returns immediately if it is not available. |
| MediaWait | MediaWait([ResIndex]) | In asynchronous mode, waits for a function to complete. |
| MediaWaitDigits | MediaWaitDigits(DigitCount [, MaxSeconds, InterDigitDelay, StopDigits]) | Waits for digits to be entered on the current Media Resource. |

#### *7.1.2.4. CTADE_A Functions. Fax Resource & Functions*

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 one Fax Resource.

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

Fax Resources are managed with the Fax functions

- **Send** faxes
- **Receive** faxes
- **Polled Transmission** or Turnarround transminision

| | | |
|---|---|---|
| FaxAbort | FaxAbort([ResIndex]) | Terminates any function in progress on a Fax Resource. |
| FaxAddRawFile | FaxAddRawFile(Filename [, Break [, Offset [, Length]]]) | Opens a raw format file in preparation for sending it with FaxSend. |
| FaxAddTextFile | FaxAddTextFile(Filename [, Break [, Offset [, Length]]]) | Opens a text file in preparation for sending it with FaxSend. |
| FaxAddTiffFile | FaxAddTiffFile(Filename [, Break [, Offset [, Length]]]) | Opens a TIFF file in preparation for sending it with FaxSend. |
| FaxClearFiles | FaxClearFiles() | Closes any open fax files. |
| FaxDisableAsync | FaxDisableAsync([ResIndex]) | Disables asynchronous mode for a Fax Resource. |
| FaxEnableAsync | FaxEnableAsync([ResIndex]) | Enables asynchronous mode for a Fax Resource. |
| FaxGetBool | BoolValue = FaxGetBool(ParmID) | Returns the Boolean value of the specified Topaz fax parameter. |
| FaxGetCount | Count = FaxGetCount([TopazTasks]) | Returns the number of Fax Resources in Topaz, assigned to the current task, or assigned to another task. |
| FaxGetIndex | ResIndex = FaxGetIndex() | Returns the ResIndex number for the current Fax Resource. |
| FaxGetInt | IntValue = FaxGetInt(ParmID[, Arg1, Arg2, ... | Returns the integer value of the specified |

| | | |
|---|---|---|
| | ArgN]) | Topaz fax parameter. |
| FaxGetOwnerTask | TaskID = FaxGetOwnerTask(ResIndex) | Returns the task ID of the task in which the specified Fax Resource is open. |
| FaxGetPageCount | Count = FaxGetPageCount() | Returns the number of pages transferred on the current Fax Resource. |
| FaxGetRemoteID | RemoteId = FaxGetRemoteID() | Returns the remote ID string received from the remote fax Resource. |
| FaxGetState | State = FaxGetState([Type]) | Returns the current state of the current Fax Resource. |
| FaxGetString | StrValue = FaxGetString(ParmID) | Returns the string value of the specified Topaz fax parameter. |
| FaxGetTech | TechType = FaxGetTech([Type]) | Returns information on the technology used by the current Fax Resource. |
| FaxGetTechIndex | TechIndex = FaxGetTechIndex() | Returns the technology index for the current Fax Resource. |
| FaxIsPollingRequested | Support = FaxIsPollingRequested() | Determines if polling has been requested. |
| FaxIsPollingSupported | Support = FaxIsPollingSupported() | Determines if polling is supported on the current Fax Resource. |
| FaxListenTo | FaxListenTo(ResType, ResIndex) | Routes the current Fax Resource to another Resource. |
| FaxReceive | FaxReceive(Filename [, Poll [, Initiate]) | Receives a fax on the current Fax Resource. |
| FaxReset | FaxReset([ResIndex]) | Resets a Fax Resource. |
| FaxSend | FaxSend([Poll, HiRes]) | Sends a fax on the current Fax Resource. |

| | | |
|---|---|---|
| FaxSetBool | FaxSetBool(ParmID, Value) | Sets the specified Topaz fax parameter to a Boolean value. |
| FaxSetHeaderText | FaxSetHeaderText(Text) | Sets the header text for faxes that will be sent. |
| FaxSetInt | FaxSetInt(ParmID, [Arg1, Arg2, ... ArgN,] Value) | Sets the specified Topaz fax parameter to an integer value. |
| FaxSetLocalID | FaxSetLocalID(LocalId) | Sets the local ID for the current Fax Resource. |
| FaxSetString | FaxSetString(ParmID, Value) | Sets the specified fax parameter to a string value. |
| FaxUnlisten | FaxUnlisten() | Disconnects a listen connection |
| FaxUnUse | FaxUnUse([ResIndex]) | Frees the current Fax Resource so that it can be used by other tasks. |
| FaxUse | FaxUse(ResIndex) | Reserves a Fax Resource for the task. |
| FaxUseNext | FaxUseNext() | Reserves the next available Fax Resource for the task. |
| FaxUseNoWait | FaxUseNoWait([ResIndex]) | Reserves a Fax Resource for the task and returns immediately if it is not available. |
| FaxWait | FaxWait([ResIndex]) | In asynchronous mode, waits for a function to complete. |

### *7.1.2.5. CTADE_A Functions. Conference Functions*

Conference Resources let you create and manage conferences.

The following topics describe the conferencing functions:

- Creating Conferences
- Removing Parties from a Conference
- Destroying Conferences

The multiconferencing functions are:

| | | |
|---|---|---|
| ConfAddParty | ConfAddParty([MaxTalkListen, MaxListenOnly, CurrentRes, Attributes]) | Adds a Resource to a Conference. |
| ConfCreate | ConfCreate(MaxTalk, MaxListen) | Reserves a Conference Resource. |
| ConfDestroy | ConfDestroy() | Removes all parties and frees all Conference Resources. |
| ConfDisableAsync | ConfDisableAsync([ResIndex]) | Disables asynchronous mode for a Conference Resource. |
| ConfEnableAsync | ConfEnableAsync([ResIndex]) | Enables asynchronous mode for a Conference Resource. |
| ConfGetBool | BoolValue = ConfGetBool(ParmID) | Returns the Boolean value of the specified Topaz Conference parameter. |
| ConfGetCount | Count = ConfGetCount([TopazTasks]) | Returns the number of Conference Resources in Topaz, assigned to the current task, or assigned to another task. |

| | | |
|---|---|---|
| ConfGetIndex | ResIndex = ConfGetIndex() | Returns the Resource index number for the current Conference Resource. |
| ConfGetInt | IntValue = ConfGetInt(ParmID [, Arg1, Arg2, ... ArgN]) | Returns the integer value of the specified Topaz Conference parameter. |
| ConfGetOwnerTask | TaskID = ConfGetOwnerTask(ResIndex) | Returns the task ID of the task in which the specified Conference Resource is open. |
| ConfGetPartyCount | PartyCount = ConfGetPartyCount() | Returns the number of parties in the current Conference. |
| ConfGetPartyResIndex | ResIndex = ConfGetPartyResIndex(PartyIndex) | Returns the Resource Index for a party in the current Conference. |
| ConfGetPartyResType | ResType = ConfGetPartyResType(PartyIndex) | Returns the Resource Type for a party in the current Conference. |
| ConfGetState | State = ConfGetState([Type]) | Returns the current state of the current Conference Resource. |
| ConfGetString | StrValue = ConfGetString(ParmID) | Returns the string value of the specified Topaz Conference parameter. |
| ConfGetTech | TechType = ConfGetTech([Type]) | Returns information on the technology used by the current Conference Resource. |
| ConfGetTechIndex | TechIndex = ConfGetTechIndex() | Returns the technology index for the current Conference Resource. |
| ConfRemoveAllParties | ConfRemoveAllParties() | Removes all parties from the Conference Resources, but keeps conference resources reserved. |
| ConfRemoveParty | ConfRemoveParty([CurrentResType]) | Removes a party from the current Conference Resource. |
| ConfReset | ConfReset() | Destroys the conference and frees the Conference Resources. |

| | | |
|---|---|---|
| ConfSetBool | ConfSetBool(ParmID, Value) | Sets the specified Topaz Conference parameter to a Boolean value. |
| ConfSetInt | ConfSetInt(ParmID, [Arg1, Arg2, ... ArgN,] Value) | Sets the specified Topaz Conference parameter to an integer value. |
| ConfSetString | ConfSetString(ParmID, Value) | Sets the specified Conference parameter to a string value. |
| ConfUnUse | ConfUnUse() | Frees the current Conference Resource for use by other tasks. |
| ConfUse | `ConfUse([ResIndex])` | Reserves a Conference Resource for the task. |
| ConfUseNext | `ConfUseNext()` | Reserves the next available Conference Resource for the task. |
| ConfUseNoWait | ConfUseNoWait([ResIndex]) | Reserves a Conference Resource for the task and returns immediately if it is not available. |
| ConfWait | ConfWait([ResIndex]) | In asynchronous mode, waits for a function to complete. |

### *7.1.2.6. CTADE_A Functions. Voice recognigtion functions*

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).

You can find the number of VR Resources on your system with the VrGetCount function.

**Note:** CTADE_A 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

| | | |
|---|---|---|
| VrAbort | VrAbort([ResIndex]) | Terminates any function in progress on a Voice Recognition Resource. |
| VrDisableAsync | VrDisableAsync() | Disables asynchronous mode. |
| VrDisableWordSpotting | VrDisableWordSpotting() | Disables word spotting. |
| VrEnableAsync | VrEnableAsync() | Enables asynchronous mode. |
| VrEnableWordSpotting | VrEnableWordSpotting(Vocab) | Enables word spotting. |
| VrEndSession | VrEndSession() | Ends a voice recognition session on the current Resource. |
| VrExecuteForm | VrExecuteForm(Form) | Executes a speech recognition form. |

| | | |
|---|---|---|
| VrGetBin | BinValue = VrGetBin(ParmID, BufNr, BytesToRead) | Returns the binary value of the specified Topaz Voice Recognition parameter. |
| VrGetBool | BoolValue = VrGetBool(ParmID) | Returns the Boolean value of the specified Topaz Voice Recognition parameter. |
| VrGetCount | Count = VrGetCount([TopazTasks]) | Returns the number of Voice Recognition Resources in Topaz, assigned to the current task, or assigned to another task. |
| VrGetFieldValue | VrGetFieldValue | |
| VrGetHypoCount | HypoCount = VrGetHypoCount() | Returns the number of hypotheses for the last recognition. |
| VrGetHypoScore | Score = VrGetHypoScore(HypoIndex) | Returns the score for a hypothesis. |
| VrGetHypoStr | Text = VrGetHypoStr(HypoIndex) | Returns the string for a hypothesis. |
| VrGetIndex | Res Index = VrGetIndex() | Returns the Resource Index number for the current Voice Recognition Resource. |
| VrGetInt | `IntValue = VrGetIntParmID [, Arg1, Arg2, ... ArgN])` | Returns the integer value of the specified Topaz Voice Recognition parameter. |
| VrGetOwnerTask | TaskID = VrGetOwnerTask(ResIndex) | Returns the task ID of the task in which the specified Voice Recognition Resource is open. |
| VrGetState | State = VrGetState([Type]) | Returns the current state of the current Voice Recognition Resource. |
| VrGetString | StrValue = VrGetString(ParmID) | Returns the string value of the specified Topaz Voice Recognition parameter. |
| VrGetTech | TechType = VrGetTech([Type]) | Returns information on the technology used by the current Voice Recognition Resource. |

| | | |
|---|---|---|
| VrGetTechIndex | TechIndex = VrGetTechIndex() | Returns the Technology Index for the current Voice Recognition Resource. |
| VrGetWordCount | WordCount = VrGetWordCount(HypoIndex) | Returns the number of words in a hypothesis. |
| VrGetWordId | WordId = VrGetWordId(HypoIndex, WordIndex) | Returns the identification number for a word in a hypothesis. |
| VrGetWordScore | Score = VrGetWordScore(HypoIndex, WordIndex) | Returns the score for a word in a hypothesis. |
| VrGetWordStr | Text = VrGetWordStr(HypoIndex, WordIndex) | Returns the text for a word in a hypothesis. |
| VrIsWordScoreSupported | Support = VrIsWordScoreSupported() | Checks whether word scores are supported by the current Voice Recognition Resource. |
| VrListenTo | VrListenTo(ResType, ResIndex) | Routes the current Voice Recognition Resource to another Resource. |
| VrPlayAndRecogAlpha | VrPlayAndRecogAlpha(FileName) | Plays a sound file and then recognizes the caller's reply, a spoken letter of the alphabet. |
| VrPlayAndRecogAlphaNum | VrPlayAndRecogAlphaNum(FileName [, MinLength] [, MaxLength]) | Plays a sound file and then recognizes the caller's reply, a combination of letters and numbers. |
| VrPlayAndRecogDate | VrPlayAndRecogDate(FileName) | Plays a sound file and then recognizes the caller's reply, a date. |
| VrPlayAndRecogDigit | VrPlayAndRecogDigit(FileName) | Plays a sound file and then recognizes the caller's reply, a single digit. |
| VrPlayAndRecogDigits | VrPlayAndRecogDigits(FileName, MinDigits, MaxDigits) | Plays a sound file and then recognizes the caller's reply, one or more digits. |
| VrPlayAndRecogGrammar | VrPlayAndRecogGrammar(FileName, GrammarFile) | Plays a sound file and then recognizes the caller's reply using the specified grammar. |
| VrPlayAndRecogInt | VrPlayAndRecogInt(FileName) | Plays a sound file and then recognizes the caller's reply, an integer. |

| | | |
|---|---|---|
| VrPlayAndRecogMoney | VrPlayAndRecogMoney(FileName) | Plays a sound file and then recognizes the caller's reply, a money value. |
| VrPlayAndRecogPhoneNumber | VrPlayAndRecogPhoneNumber(FileName) | Plays a sound file and then recognizes the caller's reply, a phone number. |
| VrPlayAndRecogTime | VrPlayAndRecogTime(FileName) | Plays a sound file and then recognizes the caller's reply, a time. |
| VrPlayAndRecogUser | VrPlayAndRecogUser(FileName, RecogType) | Plays a sound file and then recognizes the caller's reply using a user-defined recognition type. |
| VrPlayAndRecogYesNo | VrPlayAndRecogYesNo(FileName) | Plays a sound file and then recognizes the caller's reply, either "yes" or "no." |
| VrReset | VrReset(ResIndex) | Resets a Voice Recognition Resource. |
| VrSetBin | VrSetBin(ParmID, BufNr, BytesToWrite) | Sets the specified Topaz Voice Recognition parameter to a binary value. |
| VrSetBool | `VrSetBool(ParmID, Value)` | Sets the specified Topaz Voice Recognition parameter to a Boolean value. |
| VrSetInt | `VrSetInt(ParmID, [Arg1, Arg2, ... ArgN,] Value)` | Sets the specified Topaz Voice Recognition parameter to an integer value. |
| VrSetString | `VrSetString(ParmID, Value)` | Sets the specified Voice Recognition parameter to a string value. |
| VrStartSession | VrStartSession() | Starts a voice recognition session. |
| VrUnlisten | VrUnlisten() | Disconnects a listen connection. |
| VrUnUse | `VrUnUse()` | Frees the current Voice Recognition Resource for so that it can be used by other tasks. |

| | | |
|---|---|---|
| VrUse | `VrUse([ResIndex])` | Reserves a Voice Recognition Resource for the task. |
| VrUseNext | VrUseNext() | Reserves the next available Voice Recognition Resource for the task. |
| VrUseNoWait | VrUseNoWait([ResIndex]) | Reserves a Voice Recognition Resource for the task and returns immediately if it is not available. |
| VrWait | VrWait([ResIndex]) | In asynchronous mode, waits for a function to complete. |

### *7.1.3. CTADE_A. Routing*

Many telephony operations require two Resources working together. For example, playing a speech file to a caller requires a Media Resource to play the file and a Trunk Resource to handle the connection to the caller. The Trunk Resource has to listen to the Media Resource's output so the caller can hear the file, and the Media Resource should listen to the Trunk Resource in order to process any digits the caller enters.

In most cases, you won't have to explicitly control routing: by default, CTADE_A routes together the Resources you'll need to use the functions you use. For example, when you play a file, CTADE_A makes sure the current Media Resource is routed to the current Trunk Resource.

You can, however, override Topaz's automatic routing.

Some examples:

- The **MediaUse** function reserves a Media Resource and routes it to the current Trunk Resource, if there is one.

- By default, the **FaxUse** function reserves a Fax Resource that is compatible with the current Trunk Resource and routes the two together.

- If you use **MediaPlayFile** and you haven't already opened a Media Resource, Topaz reserves a Media Resource and routes it to the current Trunk Resource before playing the file. If there aren't any Trunk Resources reserved, Topaz reserves one before reserving the Media Resource.

- When you add any Resource to a Conference, Topaz routes both Resources toger.

However, it is possible to deactivate the automatic routing

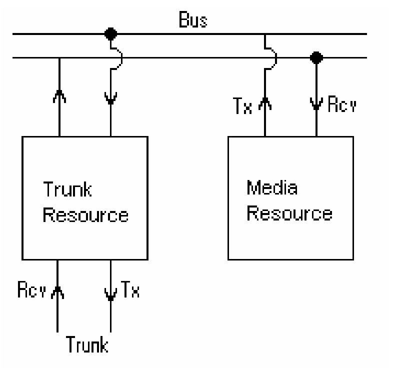
**TopazDisableAutoRoute();**

It is possible to activate again the automatic routing calling the function:

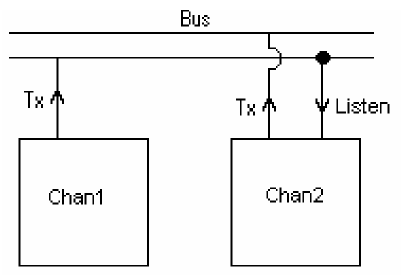
**TopazEnableAutoRoute();**

### *7.1.4. CTADE_A. Routing types: Full Duplex y Half Duplex.*

When a full-duplex connection is made, there is a bi-directional routing between two Resources, as shown in the following diagram. Each Resource listens to the other Resource's transmission, as in the typical routing of a Media and Trunk Device to each other:



However, in a half-duplex only one resource is listening to the other resource.



A full-duplex connection **TopazRoute(Res1, Index1, Res2, Index2)** makes the two routings shown by large dots in the first diagram. Resource 2 listens to Resource 1, and vice versa.

A full-duplex routing is equivalent to two listen commands.

A half-duplex connection makes the routing shown by a large dot in the above diagram. Resource 2 listens to Resource 1, but not vice versa (unless a previous routing command was made to do this).

## *7.1.5. CTADE_A. Call Control, Hang-up Processing and Onhangup*

There are 2 important questions in the hangup process:

- Detect when a caller has hangup the call
- Reacting to the hang-up when it happens.

### *7.1.5.1. Detecting hang up*

Hang-up detection is one area in which you can take advantage of Topaz's API transparency. When you write your application using the Trunk functions, you don't need to worry about the technical details of how CTADE_A will detect a hang-up.

When you deploy your application, you can use the same VOS code regardless of the system by changing configuration details in the Topaz Profile.

*7.1.5.2. Reacting to Hang-ups*

When a caller hang-up is detected, VOS stops execution at the end of the current p-code instruction (which may be in the middle of an expression or statement) and transfers control to the onhangup function. The onhangup function then executes until a restart, return or endonhangup statement is encountered. If endonhangup is encountered, the effect is equivalent to return. If a return is executed from within onhangup, execution returns to the statement that was originally interrupted. If a second hang-up event is reported while within onhangup, it is ignored.

For most applications, Parity Software recommends using restart to return control back to the beginning of the program. This usually results in the cleanest architecture.

If an ansynchronous CTADE_A function, such as MediaPlayFile, MediaRecordFile or MediaWaitDigits, is in progress when a hang-up is reported, VOS will interrupt the function and jump to onhangup immediately.

If any other blocking function is in progress when a hang-up event is reported, for example sem_set or ser_rdbuf, then VOS will wait until the function completes before jumping to onhangup.

The **TrunkDeferOnHangup and TrunkClearDeferOnHangup** functions allow an application to define critical areas of code that must not be interrupted by a jump to onhangup. A typical example might be a set of database updates which, once started, must all be completed. Any telephony blocking functions that follow TrunkDeferOnHangup will be aborted when a hang-up event is reported, but the transfer of control to the **onhangup** function will be deferred until the code reaches a TrunkClearDeferOnHangup function.

Sample:

```
dec
    var res:5;
enddec
program
  res = arg(1);
  TrunkUse(res);
  TrunkWaitCall();
…
…
  TrunkDisconnect();
  restart;
endprogram


onhangup

     TrunkDisconnect(),

     restart;

endonhangup
```

### *7.1.6. CTADE_Architecture. Technologies*

**CTADE_Architecture(Topaz)** was designed for API transparency--the same set of functions work under all supported telephony APIs and all supported trunk types--but some features are available only for 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 VOS "TrunkGetBilling" function, which couldn't be ported from the R4GcTrunk technology to other technologies, would not be API transparent.

VOS provides functions that let you control technology-specific aspects of your telephony Resources. The Get/Set functions, like TrunkGetInt, let you access technology-specific parameters and API-specific functions.

The Get/Set functions let you read and set CTADE_A parameters that apply only to particular technologies.

The Get functions, like MediaGetInt, return values for technology-level Boolean, string, and integer parameters. For example, to get the current play volume on an R4DxMedia Resource, you would use the MediaGetInt function with the REGID_PlaySpeed RegID (702):

```
Value = MediaGetInt(702);
```

The Set functions, like MediaSetInt, let you set technology-specific parameters. For example, to set the current play volume on an R4DxMedia Resource to 7, you would use the Media SetInt function with REGID_PlaySpeed (702):

```
MediaSetInt(702, 7);
```

**Note**

It's important to note that setting a CTADE_A parameter calls an API-level function that might do more than simply change a board-level parameter. For example, in R4DxMedia, you clear Dialogic tone IDs by setting REGID_R4DxMediaDisableAllToneDetection (280) to either 0 or 1 with the Media SetBool function (MediaSetBool):

**MediaSetBool(280, 0);**

Here the Media SetBool function doesn't set a board-level parameter to 0; this function calls Dialogic's dx_deltones function, which removes all user-defined tones previously added to the channel.

For details on the functions or methods available in VOS and more examples, see Get/Set Functions.

You'll find a list of available RegIDs in CTADE_A RegIDs.

The current CTADE_A technologies are:

## Trunk Technologies

| Technology | ID Number | Description |
|---|---|---|
| SimTrunk | 1001 | SimPhone Trunk |

| | | |
|---|---|---|
| R4AgTrunk | 1002 | R4 ag_ / dx_ API (aka LSI) |
| R4GcTrunk | 1003 | R4 gc_ API |
| S100Trunk | 1005 | S.100 CTscr_ API |
| R4MsTrunk | 1006 | R4 ms_ station API |

## Media Technologies

| Technology | ID Number | Description |
|---|---|---|
| SimMedia | 2001 | SimPhone Media |
| R4DxMedia | 2002 | R4 dx_ API (aka VOX) |
| S100Media | 2003 | S.100 CTplyr_/CTrcdr_ API |
| WaveMedia | 2004 | Win32 Wave API |

## Fax Technologies

| Technology | ID Number | Description |
|---|---|---|
| R4FxFax | 3001 | R4 fx_ API |
| R4GrtFax | 3002 | GammaLink fax |
| S100Fax | 3003 | S.100 Fax |

## Conference Technologies

| Technology | ID Number | Description |
|---|---|---|
| R4MsConf | 5001 | R4 ms_ conference API |
| R4DcbConf | 5002 | R4 dcb_ API |
| S100Conf | 5003 | S.100 CTconf_ API |

For instance, let's see some [3]of the RegIds under the **R4DxMedia** technology:

### R4DxMedia RegIDs

| Type | Name | Value(RegID) | Description |
|---|---|---|---|
| SetInt | ScBusListenSlot | 1000 | Sets the SC bus timeslot to listen to for this Resource (SC Bus Listen Bus Types only). |
| GetInt | ScBusTransmitSlot | 1001 | Returns the SC bus transmit timeslot for this Resource (SC Bus Listen Bus Types only). |
| SetBool | EnableDevice | 1202 | Set to false to close all device handles so a HotSwap can be done. Set to true to reopen all device handles after a HotSwap has been completed. |
| GetInt | R4DxMediaCPAConnectionType | 200 | Use this REGID to directly access the ATDX_CONNTYPE API function. |
| GetInt | R4DxMediaCPAAnswerSize | 201 | Use this REGID to directly access the ATDX_ANSRSIZ API function. |
| GetInt | R4DxMediaCPAError | 202 | Use this REGID to directly access the ATDX_CPboolOR API function. |
| GetInt | R4DxMediaCPATerm | 203 | Use this REGID to directly access the ATDX_CPTERM API function. |

[3] There are more RegIds for this tech. Please refer to the Graphical VOS VOS User's guide.

| | | | |
|---|---|---|---|
| … | … | .. | . |
| … | …... | | |

### 7.1.7. *CTADE_A Profile Ids*

#### Profile Ids

Some RegIDs describe system details and are stored in the Topaz Profile--these RegIDs are called **Profile IDs**. The contents of the Profile can be viewed with the TopazProfile.exe -L command, which generates an include file. Each entry in the generated file represents a Profile ID.

You can get the current value of a Profile ID with the appropriate Get function. For example, 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 **Topaz Profile** is a database that is quite similar to the Windows registry--it's a tree of directories and files that stores the following information:

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

Running VOS applications treat the Profile as read-only. The Profile should be fully initialized before these applications are started.

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

Entries in the Profile, called Profile IDs, 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 ID much used by Topaz code internally is

**\Techs\TechCount**

The value of TechCount is the number of different CTADE_A Technologies installed in this PC (a Technology is a specific hardware + API combination, for example Dialogic R4 VOX).

Values are one of three types: integer, string, or Boolean (True / False).

You will often see the value name and value like this:

```
\Techs\TechCount=4
```

A Topaz directory may contain values such as TechCount, or may contain an array of values all with the same name but a different integer index. (Topaz design guidelines forbid having both arrays and non-array values in the same directory.) For example, \TechTypes contains an array of integer values \TechTypes[0], \TechTypes[1] ... up to a maximum index TechCount–1.

Arrays may contain consecutive indexes as in this example, or may be "sparse", meaning that there may be gaps in the indexes so that (say) only A[1], A[16] and A[19] have values. If an array is sparse, CTADE_A requires that there is another, consecutive array which lists the indexes which have values in the sparse array. To continue the same example, there could be another array B with values B[0]=1, B[1]=16 and B[2]=19. Design guideline: CTADE_A code should never have to query a directory to find what value names are present, hence the need for the second array and for a value such as TechCount which specifies the size of the array. This helps maintain forward- and backward-compatibility and improves robustness against changes in the Resource scanner.

#### *7.1.7.1. Updating Topaz Profile*

The TopazProfile Database is update using this program:

**C:\Program Files\Parity Software\Common\Topaz\Bin\TopazProfile.exe**

TopazProfile.exe is a Win32 console application with the follows command line options:

- -S Scan devices (Profile is deleted and re-built)
- -L List Profile to "TopazProfile.txt"
- -D Delete Profile
- -C <file> Copies keys from file to Profile
- -I Update Profile as for -C, filename(s) are taken from the [ProfileIncludeFiles] section of "TOPAZ.INI"
- -F <file> Reads file, lists to standard output
- -K Keys shown as integers when listing (default is symbolic names)

## *Creating and Configuring the Topaz Profile*

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 Topaz only.

The Topaz Profile is created by running the **TopazProfile** program after completing the configuration steps below.

By default the Topaz Profile is installed on the host machine's system drive in

**C:\Program Files\Parity Software\Common\Topaz\Profile**

In order for Topaz to run, the Topaz.ini [Profile] section must specify the path to the Topaz Profile.

***"You configure certain Resource parameters in the Topaz Profile by making appropriate entries in the Topaz Configuration File (Topaz.ini) or in Profile include files."***

## RegIds and functions Get/Set

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.

The RegIDs listed in the following topics are available for the Get/Set functions. You'll also find more descriptions of Technology-specific RegIDs in topics devoted to each Technology.

Let see the RegsIds by technology:

*7.1.7.2. Topaz RegIDs: R4AgTrunk*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetInt | ScBusListenSlot | 1000 | Listen to this ScBus slot |
| SetInt | OnHookDelayMs | 2301 | On-hook delay (x10 ms) |
| SetIntAPI | R4AgSetParm | 2300 | API dx_setparm |
| GetInt | ScBusTransmitSlot | 1001 | SC bus transmit time-slot |
| GetInt | R4AgDevHandle | 2302 | Device handle from dx_open() |

*7.1.7.3.*

### *7.1.7.4. Topaz RegIDs: R4DxMedia*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetBool | AudioEventEnable | 704 | Enable Silence on/off events |
| SetBool | RecordBeep | 703 | Enable or disable the record beep. |
| SetBool | R4DxMediaDisableAllToneDetection | 280 | Removes defined tones & disables PCA, param ignored. |
| SetBool | R4DxMediaInitPerfectCallProgress | 278 | Initializes Perfect Call Analysis, param ignored. |
| SetBool | R4DxMediaEnableDialAnalysis | 285 | Enable Call Progress Analysis R4DxMediaDial |
| GetBool | RecordBeep | 703 | Record beep enabled or disabled. |
| GetIntAPI | R4DxGetParm | 221 | API dx_getparm |
| GetInt | R4DxMediaCPAConnectionType | 200 | API ATDX_CONNTYPE |
| GetInt | R4DxMediaCPAAnswerSize | 201 | API ATDX_ANSRSIZ |
| GetInt | R4DxMediaCPAError | 202 | API ATDX_CPERROR |
| GetInt | R4DxMediaCPATerm | 203 | API ATDX_CPTERM |
| GetInt | R4DxMediaCPAToneId | 204 | API ATDX_CRTNID |
| GetInt | R4DxMediaCPAFailedDialToneId | 205 | API ATDX_DTNFAIL |
| GetInt | R4DxMediaCPAFreq1Duration | 206 | API ATDX_FRQDUR |
| GetInt | R4DxMediaCPAFreq2Duration | 207 | API ATDX_FRQDUR2 |
| GetInt | R4DxMediaCPAFreq3Duration | 208 | API ATDX_FRQDUR3 |
| GetInt | R4DxMediaCPAFreq1Hz | 209 | API ATDX_FRQHZ |
| GetInt | R4DxMediaCPAFreq2Hz | 210 | API ATDX_FRQHZ2 |
| GetInt | R4DxMediaCPAFreq3Hz | 211 | API ATDX_FRQDUR3 |
| GetInt | R4DxMediaCPALongLowDuration | 212 | API ATDX_LONGLOW |
| GetInt | R4DxMediaCPAFreqOutOfBoundsPct | 213 | API ATDX_FRQOUT |
| GetInt | R4DxMediaCPAShortLowDuration | 214 | API ATDX_SHORTLOW |
| GetInt | R4DxMediaCPANonSilenceDuration | 215 | API ATDX_SIZEHI |
| GetInt | R4DxMediaCPALineStateMask | 216 | API ATDX_LINEST |

| | | | |
|---|---|---|---|
| GetInt | R4DxMediaCPAState | 217 | API ATDX_STATE |
| GetInt | R4DxMediaCPATerminationMask | 218 | API ATDX_TERMMSK |
| GetInt | R4DxMediaCPABytesTransferred | 219 | API ATDX_TRCOUNT |
| GetInt | PlaySpeed | 702 | Play speed (-10 .. 10) |
| GetInt | PlayVolume | 701 | Play volume (-10 .. 10) |
| GetInt | ScBusTransmitSlot | 1001 | SC bus transmit time-slot |
| GetInt | R4DxDevHandle | 223 | Device handle from dx_open() |
| SetInt | ClearSpeedVolumeDigits | 709 | Delete speed / volume adjustment digits |
| SetInt | PlaySpeed | 702 | Play speed (-10 .. 10) |
| SetInt | PlayVolume | 701 | Play volume (-10 .. 10) |
| SetIntAPI | R4DxSetParm | 220 | API dx_setparm |
| SetInt1 | R4DxVolumeDigit | 710 | API dx_addspddig<br>Additional parameter: Digit. |
| SetInt1 | R4DxSpeedDigit | 711 | API dx_addvoldig<br>Additional parameter: Digit. |
| SetInt | ScBusListenSlot | 1000 | Listen to this ScBus slot |
| SetInt | R4DxMediaCAP_nbrdna | 224 | # of rings before no answer. |
| SetInt | R4DxMediaCAP_stdely | 225 | Delay after dialing before analysis. |
| SetInt | R4DxMediaCAP_cnosig | 226 | Duration of no signal time out delay. |
| SetInt | R4DxMediaCAP_lcdly | 227 | Delay after dial before lc drop connect |
| SetInt | R4DxMediaCAP_lcdly1 | 228 | Delay after lc drop con. before msg. |
| SetInt | R4DxMediaCAP_hedge | 229 | Edge of answer to send connect message. |
| SetInt | R4DxMediaCAP_cnosil | 230 | Initial continuous noise timeout delay. |
| SetInt | R4DxMediaCAP_lo1tola | 231 | % acceptable pos. dev of short low sig. |
| SetInt | R4DxMediaCAP_lo1tolb | 232 | % acceptable neg. dev of short low sig. |
| SetInt | R4DxMediaCAP_lo2tola | 233 | % acceptable pos. dev of long low sig. |
| SetInt | R4DxMediaCAP_lo2tolb | 234 | % acceptable neg. dev of long low sig. |
| SetInt | R4DxMediaCAP_hi1tola | 235 | % acceptable pos. dev of high signal. |
| SetInt | R4DxMediaCAP_hi1tolb | 236 | % acceptable neg. dev of high signal. |
| SetInt | R4DxMediaCAP_lo1bmax | 237 | Maximum interval for shrt low for busy. |
| SetInt | R4DxMediaCAP_lo2bmax | 238 | Maximum interval for long low for busy. |

| | | | |
|---|---|---|---|
| SetInt | R4DxMediaCAP_hi1bmax | 239 | Maximum interval for 1st high for busy |
| SetInt | R4DxMediaCAP_nsbusy | 240 | Num. of highs after nbrdna busy check. |
| SetInt | R4DxMediaCAP_logltch | 241 | Silence deglitch duration. |
| SetInt | R4DxMediaCAP_higltch | 242 | Non-silence deglitch duration. |
| SetInt | R4DxMediaCAP_lo1rmax | 243 | Max. short low dur. of double ring. |
| SetInt | R4DxMediaCAP_lo2rmin | 244 | Min. long low dur. of double ring. |
| SetInt | R4DxMediaCAP_intflg | 245 | Operator intercept mode. |
| SetInt | R4DxMediaCAP_intfltr | 246 | Minimum signal to qualify freq. detect. |
| SetInt | R4DxMediaCAP_hisiz | 247 | Used to determine which lowmax to use. |
| SetInt | R4DxMediaCAP_alowmax | 248 | Max. low before con. if high >hisize. |
| SetInt | R4DxMediaCAP_blowmax | 249 | Max. low before con. if high <hisize. |
| SetInt | R4DxMediaCAP_nbrbeg | 250 | Number of rings before analysis begins. |
| SetInt | R4DxMediaCAP_hi1ceil | 251 | Maximum 2nd high dur. for a retrain. |
| SetInt | R4DxMediaCAP_lo1ceil | 252 | Maximum 1st low dur. for a retrain. |
| SetInt | R4DxMediaCAP_lowerfrq | 253 | Lower allowable frequency in hz. |
| SetInt | R4DxMediaCAP_upperfrq | 254 | Upper allowable frequency in hz. |
| SetInt | R4DxMediaCAP_timefrq | 255 | Total duration of good signal required. |
| SetInt | R4DxMediaCAP_rejctfrq | 256 | Allowable % of bad signal. |
| SetInt | R4DxMediaCAP_maxansr | 257 | Maximum duration of answer. |
| SetInt | R4DxMediaCAP_ansrdgl | 258 | Silence deglitching value for answer. |
| SetInt | R4DxMediaCAP_mxtimefrq | 259 | max time for 1st freq to remain in bounds |
| SetInt | R4DxMediaCAP_lower2frq | 260 | lower bound for second frequency |
| SetInt | R4DxMediaCAP_upper2frq | 261 | upper bound for second frequency |
| SetInt | R4DxMediaCAP_time2frq | 262 | min time for 2nd freq to remains in bounds |
| SetInt | R4DxMediaCAP_mxtime2frq | 263 | max time for 2nd freq to remain in bounds |
| SetInt | R4DxMediaCAP_lower3frq | 264 | lower bound for third frequency |
| SetInt | R4DxMediaCAP_upper3frq | 265 | upper bound for third frequency |
| SetInt | R4DxMediaCAP_time3frq | 266 | min time for 3rd freq to remains in bounds |
| SetInt | R4DxMediaCAP_mxtime3frq | 267 | max time for 3rd freq to remain in |

| | | | |
|---|---|---|---|
| | | | bounds |
| SetInt | R4DxMediaCAP_dtn_pres | 268 | Length of a valid dial tone (def=1sec) |
| SetInt | R4DxMediaCAP_dtn_npres | 269 | Max time to wait for dial tone (def=3sec) |
| SetInt | R4DxMediaCAP_dtn_deboff | 270 | The dialtone off debouncer (def=100ms) |
| SetInt | R4DxMediaCAP_pamd_failtime | 271 | Wait for AMD/PVD after cadence break(default=4sec) |
| SetInt | R4DxMediaCAP_pamd_minring | 272 | min allowable ring duration (def=1.9sec) |
| SetInt | R4DxMediaCAP_pamd_spdval | 273 | Set to 2 selects quick decision (def=1) |
| SetInt | R4DxMediaCAP_pamd_qtemp | 274 | The Qualification template to use for PAMD |
| SetInt | R4DxMediaCAP_noanswer | 275 | time before no answer after first ring (default=30sec) |
| SetInt | R4DxMediaCAP_maxintering | 276 | Max inter ring delay before connect (8 sec) |
| SetInt | R4DxMediaEnableTone | 281 | Enables an added tone. |
| SetInt | R4DxMediaDisableTone | 282 | Disables an added tone. |
| SetInt | R4DxMediaEnableEchoCancel | 286 | Enable echo cancel (dx_listenecr) |
| SetInt | R4DxMediaEnableEchoCancelNLPOn | 287 | Enable echo cancel & enable NLP (dx_listenecrex) |
| SetInt | R4DxMediaEnableEchoCancelNLPOff | 288 | Enable echo cancel & disable NLP (dx_listenecrex) |
| SetInt | R4DxMediaDisableEchoCancel | 289 | Disable echo cancel, int param ignored (dx_unlistenecr) |
| SetStr | R4DxMediaEnableToneDetection | 279 | Adds a defined tone. (i.e. '101;BUSY') |
| SetStr | R4DxMediaDefinePerfectCallProgressTone | 283 | Defines a PCA tone. (i.e. '253;BUSY') |
| SetStr | R4DxMediaDial | 284 | Performs a dx_dial regardless of current state |

#### *7.1.7.5. Topaz RegIDs: R4GcTrunk*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetBool | R4GcStoreBillingInfo | 500 | Store data from gc_GetBilling at end of call |
| SetBool | R4GcAttachSupported | 536 | Is gc_Attach supported/needed |

| | | | |
|---|---|---|---|
| SetBool | R4GcStoreAlertingInfo | 515 | Do we store alering info |
| SetBool | R4GcTerminateOnAlerting | 516 | Do we terminate the call when alerting received |
| GetIntAPI | R4GcGetParm | 507 | API gc_GetParm |
| GetIntAPI | R4GcGetEventResultIntGC | 537 | Get integer event result for Global Call |
| GetIntAPI | R4GcGetEventResultIntLIB | 539 | Get integer event result for specific GC library |
| GetIntAPI | R4GcGetLinedevState | 541 | Get line state |
| GetInt | CallHandle | 708 | Get current CRN |
| GetInt | ScBusTransmitSlot | 1001 | Get Sc Bus transmit Slot |
| GetInt | R4GcCallInfoCategoryDigit | 547 | Calling party category for the call |
| GetInt | R4GcCallInfoConnectType | 548 | Connection analysis |
| GetInt | R4GcGetCallState | 504 | Get call state |
| GetInt | R4GcGetNetCRV | 542 | Get network CRV |
| GetInt | R4GcCPAConnectionType | 519 | Get connection type CPA result |
| SetIntAPI | R4GcSetParm | 508 | API gc_SetParm |
| SetIntAPI | R4GcSetBillingInfo | 511 | API gc_SetBilling |
| SetInt | R4GcDNISDigitCount | 501 | Number of overlap DNIS digits to expect |
| SetInt | R4GcCPACallTimeout | 503 | Timeout (seconds) before reporting Ring No Answer |
| SetInt | ScBusListenSlot | 1000 | Listen to this ScBus slot |
| SetInt | R4GcAckServiceISDN | 543 | Send the first response to an incoming call |
| SetInt | RingsBeforeAnswer | 700 | How many rings before answer is reported |
| SetInt | R4GcSetSigInfoId | 555 | What sig info to get: UUI(7) or U_IES(16) |
| SetInt | R4GcSetMsgTypeId | 557 | What msg type to send, i.e. SndMsg_Congestion |
| SetInt | R4GcAcceptCall | 518 | Accept Call function |
| SetInt | R4GcMKBSetDefaults | 589 | Set MAKECALL_BLK to default values |
| SetInt | R4GcMKBXferCap | 570 | MAKECALL_BLK.BC_xfer_cap |
| SetInt | R4GcMKBXferMode | 571 | MAKECALL_BLK.BC_xfer_mode |
| SetInt | R4GcMKBXferRate | 572 | MAKECALL_BLK.BC_xfer_rate |
| SetInt | R4GcMKBUsrL1Protocol | 573 | MAKECALL_BLK.usrinfo_layer1_protocol |
| SetInt | R4GcMKBUsrRate | 574 | MAKECALL_BLK.usr_rate |
| SetInt | R4GcMKBDestNumType | 575 | MAKECALL_BLK.destination_number_type |

| | | | |
|---|---|---|---|
| SetInt | R4GcMKBDestNumPlan | 576 | MAKECALL_BLK.destination_number_plan |
| SetInt | R4GcMKBDestSubNumType | 577 | MAKECALL_BLK.destination_sub_number_type |
| SetInt | R4GcMKBDestSubNumPlan | 578 | MAKECALL_BLK.origination_sub_number_plan |
| SetInt | R4GcMKBOrigNumType | 579 | MAKECALL_BLK.origination_number_type |
| SetInt | R4GcMKBOrigNumPlan | 580 | MAKECALL_BLK.origination_number_plan |
| SetInt | R4GcMKBOrigSubNumType | 581 | MAKECALL_BLK.origination_sub_number_type |
| SetInt | R4GcMKBOrigSubNumPlan | 582 | MAKECALL_BLK.origination_sub_number_plan |
| SetInt | R4GcMKBFacilityFeatureService | 583 | MAKECALL_BLK.facility_feature_service |
| SetInt | R4GcMKBFacilityCodingValue | 584 | MAKECALL_BLK.facility_coding_value |
| SetInt | R4GcMKBCompletionPoint | 585 | MAKECALL_BLK.completion_point |
| GetStr | R4GcGetBillingInfo | 510 | Billing info |
| GetStr | R4GcGetEventResultStrGC | 538 | Get string event result for Global Call |
| GetStr | R4GcGetEventResultStrLIB | 540 | Get string event result for specific GC library |
| GetStr | R4GcCallInfoType | 546 | Charge or no charge call |
| SetStr | R4GcMKBDestSubPhoneNr | 586 | MAKECALL_BLK.destination_sub_phone_number |
| SetStr | R4GcMKBOrigPhoneNr | 587 | MAKECALL_BLK.origination_phone_number |
| SetStr | R4GcMKBOrigSubPhoneNr | 588 | MAKECALL_BLK.origination_sub_phone_number |
| SetStr | R4GcCallingAddress | 534 | Phone number of this trunk |
| SetStr | R4GcTraceFileName | 535 | API gc_StartTrace (empty string=gc_StopTrace) |
| GetBin | R4GcCallInfoU_IES | 549 | Get unformatted information elements |
| GetBin | R4GcCallInfoUUI | 550 | Get user-to-user info |
| GetBin | R4GcGetFrame | 551 | Retrieve the frame received by application |
| GetBin | R4GcGetIE | 553 | Retrieve the info element from the incoming message |
| GetBin | R4GcGetSigInfo | 556 | Get signalling info from incoming message |
| GetBin | R4GcCallInfo | 514 | Get prevoiusly stored callinfo |
| SetBin | R4GcSetIE | 554 | Set info element for outgoing message |
| SetBin | R4GcSndFrame | 552 | Send the frame |
| SetBin | R4GcSndMsg | 512 | API gc_SndMsg |

### *7.1.7.6. Topaz RegIDs: R4MsTrunk*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetBool | R4MsLoopOnIsInboundCall | 324 | Interpret loop on as seize |
| SetBool | R4MsAcceptByDialTone | 325 | Accept in-bound call by playing dial tone |
| SetBool | R4MsOutboundByRing | 326 | Make out-bound call by ringing |
| SetBool | R4MsOutboundByZipTone | 327 | Make out-bound call by playing zip tone |
| SetBool | R4MsDisconnectByBatteryOff | 328 | Disconnect by turning battery off |
| SetBool | R4MsTrunkFlashHookFlag | 332 | Set flag which 'remembers' flash-hook |
| GetBool | R4MsTrunkFlashHookFlag | 332 | Get flag which 'remembers' flash-hook |
| SetInt | R4MsTrunkWaitFlashHook | 333 | Wait flash-hook, parm is secs time-out |
| SetInt | R4MsRingCount | 321 | Max times to ring station 0..255 |
| SetInt | R4MsDisconnectMs | 329 | Battery off time for disconnect (ms) |
| SetInt | R4MsAdjustStationVolume | 322 | API ms_setvol (VOLADJ) |
| SetInt | R4MsSetStationVolume | 323 | API ms_setvol (VOLRES then VOLADJ) |
| SetInt | ScBusListenSlot | 1000 | Listen to this ScBus slot |
| SetIntAPI | R4MsSetBoardParm | 330 | Set board level parameter |
| SetIntAPI | R4MsSetStationParm | 331 | Set station level parameter |

### *7.1.7.7. Topaz RegIDs: S100Conf*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetInt | S100ConfCreateTimeout | 2500 | Maximum time to wait for availability of conference resources. |
| SetInt | S100GroupCreateTimeout | 2501 | Maximum time to wait for availability of group resources. |
| GetInt | S100ConfCreateTimeout | 2500 | Maximum time to wait for availability of conference resources. |

| | | | |
|---|---|---|---|
| GetInt | S100GroupCreateTimeout | 2501 | Maximum time to wait for availability of conference resources. |

### *7.1.7.8. Topaz RegIDs: S100Fax*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetBool | S100FaxAutoFooter | 2408 | Footer text source: if true, from fax hardware, else from application. |
| SetBool | S100FaxAutoHeader | 2409 | Header text source: if true, from fax hardware, else from application. |
| GetBool | S100FaxAutoFooter | 2408 | Footer text source: if true, from fax hardware, else from application. |
| GetBool | S100FaxAutoHeader | 2409 | Header text source: if true, from fax hardware, else from application. |
| SetInt | S100FaxFirstPageNum | 2411 | Number of the first page to be transmitted. |
| SetInt | S100FaxFooterLength | 2413 | Maximum number of characters in the page footer. |
| SetInt | S100FaxHeaderLength | 2416 | Maximum number of characters in the page header. |
| GetInt | S100FaxScanTime | 2403 | Negotiated scan time in milliseconds. |
| GetInt | S100FaxTransferSpeed | 2404 | Negotiated transfer speed in bits per second. |
| GetInt | S100FaxSMPageNum | 2405 | The page of SM Data object that was transferred. |
| GetInt | S100FaxHeaderNum | 2406 | The latest page number on the fax header. |
| GetInt | S100FaxPagesTransferred | 2407 | The number of pages transferred. |
| GetInt | S100FaxFirstPageNum | 2411 | Number of the first page to be transmitted. |
| GetInt | S100FaxFooterLength | 2413 | Maximum Number of characters in the page footer. |
| GetInt | S100FaxFooterPlacement | 2414 | Selects the way footers are placed into the transmitted image. |
| GetInt | S100FaxHeaderLength | 2416 | Maximum number of characters in the page header. |
| GetInt | S100FaxHeaderPlacement | 2417 | Selects the way headers are placed into the transmitted image. |
| GetInt | S100FaxPageWidth | 2418 | Maximum width of a transmitted page in Pixels. |
| SetStr | S100FaxFaxID | 2410 | Fax ID of local fax. |
| SetStr | S100FaxFooterField | 2412 | Text for fax footer field. |
| SetStr | S100FaxHeaderField | 2415 | Text for fax header field. |

| | | | |
|---|---|---|---|
| GetStr | S100FaxRemoteID | 2401 | Fax ID of remote fax machine. |
| GetStr | S100FaxFaxID | 2410 | Fax ID of local fax. |
| GetStr | S100FaxFooterField | 2412 | Text for fax footer field. |
| GetStr | S100FaxHeaderField | 2415 | Text for fax header field. |

#### *7.1.7.9. Topaz RegIDs: S100Media*

| ***Type*** | ***RegID*** | ***RegID*** | ***Description*** |
|---|---|---|---|
| SetBool | S100RecorderStartBeep | 2600 | Precede record with a beep. |
| SetBool | S100RecorderPauseCompressionOn | 2602 | Remove speech pauses from recording if true. |
| GetBool | S100RecorderStartBeep | 2600 | Precede record with a beep. |
| GetBool | S100RecorderPauseCompressionOn | 2602 | Remove speech pauses from recording if true. |
| SetInt | S100RecorderBeepFrequency | 2604 | Frequency of record start beep in Hertz. |
| SetInt | S100RecorderBeepLength | 2605 | Length of record start beep in milliseconds. |
| SetInt | S100RecorderCoder | 2606 | Coder type for record. |
| SetInt | S100RecorderMinDuration | 2608 | Minimum duration of record in milliseconds. |
| SetInt | S100RecorderPauseThreshold | 2609 | The threshold time in milliseconds for which pause compression is triggered. |
| SetInt | S100RecorderSilenceThreshold | 2610 | The threshold time in milliseconds for which silence termination is triggered. |
| GetInt | S100RecorderBeepFrequency | 2604 | Frequency of record start beep in Hertz. |
| GetInt | S100RecorderBeepLength | 2605 | Length of record start beep in milliseconds. |
| GetInt | S100RecorderCoder | 2606 | Coder type for record. |
| GetInt | S100RecorderMinDuration | 2608 | Minimum duration of record in milliseconds. |
| GetInt | S100RecorderPauseThreshold | 2609 | The threshold time in milliseconds for which pause compression is triggered. |
| GetInt | S100RecorderSilenceThreshold | 2610 | The threshold time in milliseconds for which silence termination is triggered. |
| GetStr | S100CurrentContainer | 2700 | Current container. |

| | | | |
|---|---|---|---|
| SetStr | S100CurrentContainer | 2700 | Current container. |

### *7.1.8. CTADE_A. Configuration. TOPAZ.INI*

The Topaz.ini file is used to specify various configuration parameters that are read by the Topaz engine when it is first invoked on your system. If you change any entries in Topaz.ini, you must be sure that all instances of VOS and CallSuite, which use Topaz, are closed before the changes can take effect.

| **Sección** | **Descripción** |
|---|---|
| [Profile] | Topaz Profile startup information. |
| [ProfileIncludeFiles] | Include files to be merged with the Topaz Profile during AutoMerge |
| [ProfileExcludeTechnologies] | Telephony technologies to exclude from the Topaz Profile scan |
| [ProfileDependencies] | Windows NT/2000 services that must be started before the TopazProfile.exe utility will start.. |
| [SimMediaScanner] | IDs of the Wave In and Wave Out devices you want SimMedia to use. |

|  |  |
|---|---|
| [WaveScanner] | IDs of the Wave In and Wave Out devices you want WaveMedia to use. |
| [S100Groups] | S100 groups and their corresponding Topaz Resources that will be used by the application. |
| [S100Defaults] | Defaults for keys in the [S100Groups] section. |

### *7.1.8.1. [Profile] Section*

The Profile section of Topaz.ini lets you set parameters that control the Topaz Profile at startup.

```
[Profile]
Path=e:\Program Files\Parity Software\Common\Topaz\Profile
```

### *7.1.8.2. [ProfileIncludeFiles] Section*

The ProfileIncludeFiles section of Topaz.ini lists the files that are to be copied to the Topaz Profile each time the TopazProfile.exe utility runs with the Include (-I) option.

Example

```
[ProfileIncludeFiles]
C:\Topaz\LangRegs.tzp
C:\Topaz\IPFRegs.tzp
```

#### *7.1.8.3. [ProfileExcludeTechnologies] Section*

The ProfileExcludeTechnologies section of Topaz.ini lists technologies that are to be excluded from the TopazProfile.exe Autoscan.

Example

```
        [ProfileExcludeTechnologies]
        S100Trunk
        S100Media
```

#### *7.1.8.4. [ProfileDependencies] Section*

The ProfileDependencies section of Topaz.ini lists the Windows NT/2000 services that must be started before the TopazProfile.exe utility will start.

Example

```
        [ProfileDependencies]
        Alerter
        Dialogic
```

#### *7.1.8.5. [SimMediaScanner] Section*

The SimMediaScanner section of Topaz.ini specifies the IDs of the Wave In and Wave Out devices you want SimMedia to use.

```
[SimMediaScanner]
WaveIn=<Wave In ID>
WaveOut=<Wave Out ID>
```

By default, both Wave ID values are zero.

Example

```
[SimMediaScanner]
WaveIn=1
WaveOut=1
```

#### *7.1.8.6. [WaveScanner] Section*

The WaveScanner section of Topaz.ini specifies the IDs of the Wave In and Wave Out devices you want the WaveMedia technology to use.

Example

```
[WaveScanner]
WaveDeviceID0=1,1
```

*7.1.8.7. [S100Groups] Section*

The S100Groups section of Topaz.ini specifies the S100 groups and their corresponding Topaz Resources that will be used by the application.

Default settings for the S100Groups section entries are stored in the [S100Defaults] Section.

Example

```
[S100Groups]
CCRGroups = 4
withMedia
ASI = CCRMEDIA_ASI
```

*7.1.8.8. [S100Defaults] Section*

The S100Defaults section of Topaz.ini specifies default values for keys in the S100Groups section. If you do not specify a default value for a key in the S100Defaults section, Topaz will use the standard default value. Valid keys and their default values are listed in the example.

Example

```
[S100Defaults]
Server = Default_Server
AppProfile = Parity_TOPAZ_ProfileS100v2
MakeCallGroupConfig =
MakeCallGroupSet =
SPRGroupConfig = SPRMEDIA_GROUPCONFIG
SPRGroupSet = SPRMEDIA_GROUPSET
SPROnlyGroupConfig = SPRONLY_GROUPCONFIG
SPROnlyGroupSet = SPRONLY_GROUPSET
```

The MakeCallGroupSet, SPROnlyGroupSet, and SPRGroupSet keys apply only to CT Media V1. Group sets were removed from CT Media Version 2.

See S100 Topaz.ini Configuration for a detailed description of this section and the S100Groups section

## *7.1.9. CTADE_A. Tones*

In this chapter, we will see how to generate and detect tones.

### *7.1.9.1. Playing Tones.*

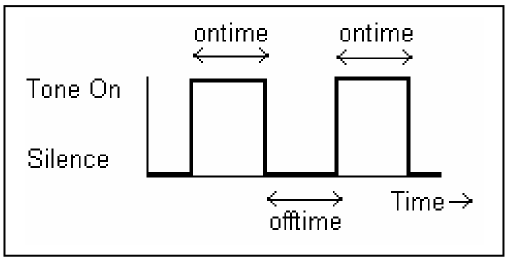
Tones, also called general tones or global tones, may be one of the following four types: single, dual, single with cadence, and dual with cadence.

#### **Single and Dual Tones**

A single tone has only one amplitude and frequency. A dual tone has two sets of amplitudes and frequencies. Most familiar tones from the public telephone network, including touch-tones, are dual tones.

#### **Cadence**

A tone with cadence has a regular pattern of periods where the tone is present and where there is silence. In the US, the busy tone is a cadence tone. A cadence pattern looks like this:



Tones without cadence are called continuous tones. For example, touch tones are continuous tones.

**Playing Tones**

You can play a single continuous tone with the **MediaPlaySingleContinuousTone** function:

**MediaPlaySingleContinuousTone**(Freq1, Amp1, Duration, StopTones);

The Freq1 parameter defines the frequency of the tone, which can be between 300 and 2000 Hz. The Amp1 parameter defines the tone's amplitude, measured in decibels. Valid amplitudes range from -40 to 2 dB. The Duration parameter specifies the length of the tone, in tenths of a second. The StopTones parameter is a string of the touch tone digits that will interrupt the playing or recording of any sound file.

To play a continuous dual tone, use the **MediaPlayDualContinuousTone** function:

**MediaPlayDualContinuousTone**(Freq1, Amp1, Freq2, Amp2, Duration, StopTones);

The parameters are almost identical to MediaPlaySingleContinuousTone, except that Freq2 and Amp2 specify the frequency and amplitude of the second component of the tone.

The functions that play cadence tones are similar to those that play continuous tones:

**MediaPlaySingleCadenceTone**(Freq1, Amp1, OnTime, OffTime, Count, StopTones);
**MediaPlayDualCadenceTone**(Freq1, Amp1, Freq2, Amp2, OnTime, OffTime, Count, StopTones);

In addition to the Freq1, Amp1, Freq2, and Amp2 parameters, which are used just as in the continuous tone functions, the OnTime parameter specifies the length of time, in tenths of a second, that the tone plays in each repetition of the cadence. The OffTime parameter defines the length of the silence, in tenths of a second, in each repetition of the cadence. The Count parameter specifies the number of times to repeat the cadence.

If you've set up the parameters for a tone in the Topaz Profile, you can play that tone on a Media Resource with the **MediaPlayTone** function:

**MediaPlayTone**(ToneName, StopTones);

The ToneName parameter must match the name of a tone you've set up in the Topaz Profile. The optional StopTones parameter sets the stop tones.

*7.1.9.2. Tone Include File*

The tone include file is a .TZP file that lets you specify which tones are directly supported by CTADE_A.

**Note**

In order to add information to the Topaz Profile from your tone include file, you must merge the Topaz Profile with the include file using the TopazProfile.exe Copy command (-C). To make sure that the most recent tone include file is merged with CTADE_A each time it starts, you can add a reference to the include file in the [ProfileIncludeFiles] section of the Topaz.ini file and use the TopazProfile.exe Include option (-I).

**ToneCount**

The ToneCount Profile ID specifies how many tones are to be defined. Remember that tones are numbered starting with 0.

**ToneName**

The tone name is used by your VOS application to specify which tone to play.

**ToneType**

Currently, four types of tones are supported in CTADE_A:

*SingleContinuous*

*SingleCadence*

*DualContinuous*

*DualCadence*

See Playing Tones for descriptions of single and dual, continuous and cadence tones.

**Frequencies (ToneFreq1, ToneFreq2)**

The frequency of the tone can be between 300 and 2000 Hz.

**Amplitudes (ToneAmp1, ToneAmp2)**

A tone's amplitude is measured in decibels. Valid amplitudes range from -40 to 2 dB.

**ToneOnTime**

The ToneOnTime parameter specifies the length of time, in tenths of a second, that the tone plays in each repetition of the cadence.

**ToneOffTime**

The ToneOffTime Profile ID defines the length of the silence, in tenths of a second, in each repetition of the cadence.

**ToneFreq1Var**

Variance allowed during detection of frequency 1, in Hz.

**ToneFreq2Var**

Variance allowed during detection of frequency 2, in Hz.

**ToneOnTimeVar**

Variance in the cadence "OnTime" allowed during detection of the tone, in tenths of a second.

**ToneOffTimeVar**

Variance in the cadence "OffTime" allowed during detection of the tone, in tenths of a second.

**ToneRepetitions**

Number of repetitions of the cadenced tone required before affirmative detection.

**ToneChar**

ToneChar assigns a character (from E to Z) to the tone. Before you can perform custom tone detection, you must assign a character to the tone.

**ToneIsBuffered**

ToneIsBuffered is a Boolean value that indicates whether CTADE_A will place the tone's character in the digit buffer, to be retrieved by the VOS MediaGetDigitBuffer function.

**Examples**

```
\ToneCount=5

\Tones[0]\ToneName=BUSY
\Tones[0]\ToneType=DualCadence
\Tones[0]\ToneFreq1=480
\Tones[0]\ToneAmp1=-20
\Tones[0]\ToneFreq2=620
\Tones[0]\ToneAmp2=-20
\Tones[0]\ToneOnTime=5
\Tones[0]\ToneOffTime=5

\Tones[1]\ToneName=DIAL
\Tones[1]\ToneType=DualContinuous
\Tones[1]\ToneFreq1=350
\Tones[1]\ToneAmp1=-20
\Tones[1]\ToneFreq2=440
\Tones[1]\ToneAmp2=-20

\Tones[2]\ToneName=TestTone
\Tones[2]\ToneType=DualContinuous
\Tones[2]\ToneFreq1=1209
\Tones[2]\ToneFreq2=697
\Tones[2]\ToneAmp1=-20
\Tones[2]\ToneAmp2=-20
\Tones[2]\ToneChar=T
\Tones[2]\ToneIsBuffered=True

\Tones[3]\ToneName=DISCONNECT
\Tones[3]\ToneType=DualContinuous
\Tones[3]\ToneFreq1=1477
\Tones[3]\ToneFreq2=697
\Tones[3]\ToneFreq1Var=50
\Tones[3]\ToneFreq2Var=50

\Tones[4]\ToneName=DISCONNECT2
\Tones[4]\ToneType=DualCadence
\Tones[4]\ToneFreq1=697
\Tones[4]\ToneFreq2=1477
\Tones[4]\ToneOnTime=10
\Tones[4]\ToneOffTime=10
\Tones[4]\ToneFreq1Var=50
\Tones[4]\ToneFreq2Var=50
\Tones[4]\ToneOnTimeVar=5
\Tones[4]\ToneOffTimeVar=5
\Tones[4]\ToneRepetitions=2
```

# 8. VOS Runtime-Log

The VOS Runtime is generating logs in the files **VOS1.log** and **VOS2.log**. The Latest logs is always in the file VOS1.log. The content of these files is as follow:

## *Start of run*

```
010518 172034.35 VOS loaded from C:\Program Files\Parity Software\Graphical
VOS\Bin\Vos7d.dll

010518 172034.35 VOS 7 (Debug) Built Apr  4 2001 13:43:51
```

## *End of run*

```
010518 172038.05 CASEST~1:062b[0] VOS 7 (Debug) Built Apr  4 2001 13:43:51

010518 172038.05 CASEST~1:062b[0] Stopping: exit(21)
```

## *Warnings, Failures, and Errors*

```
010518 172036.14 @W 713 RLL Adorll missing, wrong version or loaded in wrong order


010518 172036.17 CASEST~1:149b[0] @F fil_open(LOGO,r): 088 error 2


010518 172037.91 CASEST~1:0606[0] @E CADOConn 'Open' : Films : COM Error
0x80004005(Unspecified error  ) : _Connection::Open()

010518 172037.91 CASEST~1:0606[0]            Source = Microsoft OLE DB Provider for ODBC
Drivers

010518 172037.91 CASEST~1:0606[0]            Description = [Microsoft][ODBC Microsft
Access Driver] Could not find file '(unknown)'.
```

**Built-in functions returns**

## *Builtin function calls*

```
010518 172036.15 CASEST~1:004b[0] @B getpid() = 0
```

**R4 library function returns**

## *Driver / API*

```
010727 164728.19 @D ATDX_TERMMSK(1[dxxxB1C1])=0x0

010727 164728.19 @D Dev 1 dxxxB1C1 Evt 0x86 Data 0x15b9350 CST Data 0x0 Rings received

010727 164728.19 @D ATDX_TERMMSK(1[dxxxB1C1])=0x0
```

**TOPAZ Event**

## *Topaz Events / States*

```
010727 164728.19 CASEST~2:0017[1] @Y Trunk:0 Event dxxxB1C1[1] 134 CST Data 0x0 Rings
received

010727 164728.19 CASEST~2:0017[1] @Z Trunk:0 Idle->InboundRinging (1)
```

**Topaz state change**

```
010727 164728.19 ResumeTask(1, 1)

010727 164728.27 CASEST~2:001f[1] @D dx_sethook(1[dxxxB1C1], 1, EV_ASYNC)=0

010727 164728.27 CASEST~2:001f[1] @B TrunkAnswerCall() = 1
```