Developer Guide

6.0

ODT Vision®

Developer Guide

Chapter 1 - Introduction to the ODT VISION® Developer Guide

Notes and acknowledgments

Information in these manuals and features of the **ODT Vision®** system are subject to change without notice and does not represent a commitment on the part of Ohio Data Transfer. The software described in this manual are furnished under a license agreement and may only be used or copied in accordance with the terms of the agreement. It is against the law to copy the software to any medium except as specifically allowed in the license agreement. The licensee may make copies of the software for backup purposes only. No part of this manual or software may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by fax transmittal for any purpose other than the licensee's personal use, without the express written permission of Ohio Data Transfer.

All names and addresses contained in this manual and the accompanying software are part of a completely fictitious scenario, and are only included to document the use of this product.

Microsoft, the Microsoft logo, MS, MS-DOS, FoxPro, and Access are registered trademarks and Windows is a trademark of Microsoft Corporation in the USA and other countries.

IBM is a registered trademark of the International Business Machines Corporation.

TrueType is a registered trademark of Apple Computer, Inc.

Btrieve is a registered trademark of SoftCraft, Inc., a Novell company.

dBASE is a registered trademark of Ashton-Tate Corporation.

Paradox is a registered trademark of Ansa Software, a Borland company.

About this manual

This manual is designed to assist you in creating scripts that are used to control the ODT VISION® system.

What is the ODT VISION® Compiler?

Applications for the **ODT VISION®** system are created using a simple, easy to learn scripting language. Any text editor like Windows Notepad can be used to edit scripts. The default extension used for a script source file is "ODT". The **ODT VISION®** Compiler is then used to compile the scripts into a form that can be used by the **ODT VISION®** system.

If your **ODT VISION®** system came with a "turn-key" custom designed application, the use of the Compiler program may not be required.

System requirements for the Compiler

- IBM compatible PC with 500 MHz or higher processor
- VGA or higher resolution monitor
- 128 Meg of memory or higher
- Windows 98 or higher version of an Operating System.

Product support

Product technical support is available through your **ODT VISION®** representative. If you do not know the name of your **ODT VISION®** representative, contact Ohio Data Transfer. at (614) 985 – 3814 for further assistance.

Software program license and warranty

Customer Agreements

Carefully read the following terms and conditions before opening the diskette envelope. Opening the diskette envelope indicates your acceptance of these terms and conditions. If you do not agree with the terms, you should promptly return the complete **ODT VISION®** system hardware and software package including the unopened diskette envelope and all original packing material, and your money will be refunded.

Ohio Data Transfer. (hereinafter referred to as ODT) provides this software and licenses its use in the United States. You assume responsibility for the selection of the software to achieve the desired results and for the installation, use, and results obtained from the software.

License

You may use the program on any machine that you own or use and may copy the program into any machine readable or printed form for backup support of your use of the program. You may not use, copy, modify, sublicense, or otherwise transfer the software of any copy, modification or merged portion, in the whole or in part, except as expressly provided for in this license. If you transfer possession of any copy, modification, or merged portion of the program to another party, your license is automatically terminated and any attempted sublicense, assignment or other transfer is null and void.

Term

The license is effective until terminated. You may terminate it at any time by destroying the software along with all copies, modification and merged portions in any form. It will also be terminated upon conditions set forth elsewhere in this agreement of if you fail to comply with any term or condition of this agreement. You agree upon such termination to destroy the software together with all copies, modifications and merged portions in any form.

Limited Warranty

This program is provided “as is” without warranty of any kind, either expressed or implied, including but not limited to the implied warranties or merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the software is with you. Should the program prove defective, you (and not ODT nor their representatives) assumes the entire cost of all necessary servicing, repair or correction. Some states do not allow the exclusion of implied warranties, do the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights that vary from state to state. ODT does not warrant that the features contained in this software will meet your requirements nor that the operation of the software will be uninterrupted or error free. However, ODT does warrant the diskette on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

Limitations of remedies

ODT’s entire liability and your exclusive remedy shall be the replacement of any diskette not meeting with ODT’s warranty described above and which is returned to ODT with a copy of your payment receipt, or if ODT is unable to deliver a replacement diskette which is free of defects in materials or workmanship, you may terminate this agreement by returning the software and any accompanying hardware and your money will be refunded.

In no event will ODT be liable for any damages including lost profits, lost savings, or other incidental or consequential damages arising from the use or inability to use the software even if ODT has been advised of the possibility of such damages, or any claim by any other party. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

General

This agreement shall be governed by the laws of the State of Ohio. Should you have any questions concerning this agreement, you may contact ODT in writing. You acknowledge that you have read this agreement, understand it and agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive statement of agreement between you and ODT which supersedes any proposal or prior agreement, oral or written, and any other communications between you and ODT relating to the subject matter of this agreement

This page left intentionally blank

Chapter 2 - ODT VISION® Compiler

Starting and quitting the ODT VISION® Compiler

Before starting the ODT VISION® Compiler

Before starting the **ODT VISION®** Compiler, you must create at least one script source file. Sample scripts were installed automatically if you selected that option during the **ODT VISION®** software installation.

Installing the ODT VISION® Compiler

The **ODT VISION®** Compiler program was installed as part of the **ODT VISION®** system installation. Any computer other than the **ODT VISION®** main system can be used as a development machine by installing the system software. See the chapter on software installation in your **ODT VISION®** System Reference manual for more information.

Starting the ODT VISION® Compiler

From the **ODT VISION®** program group, select the Compiler option.

Using the ODT VISION® Compiler

Functions of the **ODT VISION®** Compiler can be performed from either menu options, or by using the toolbar.

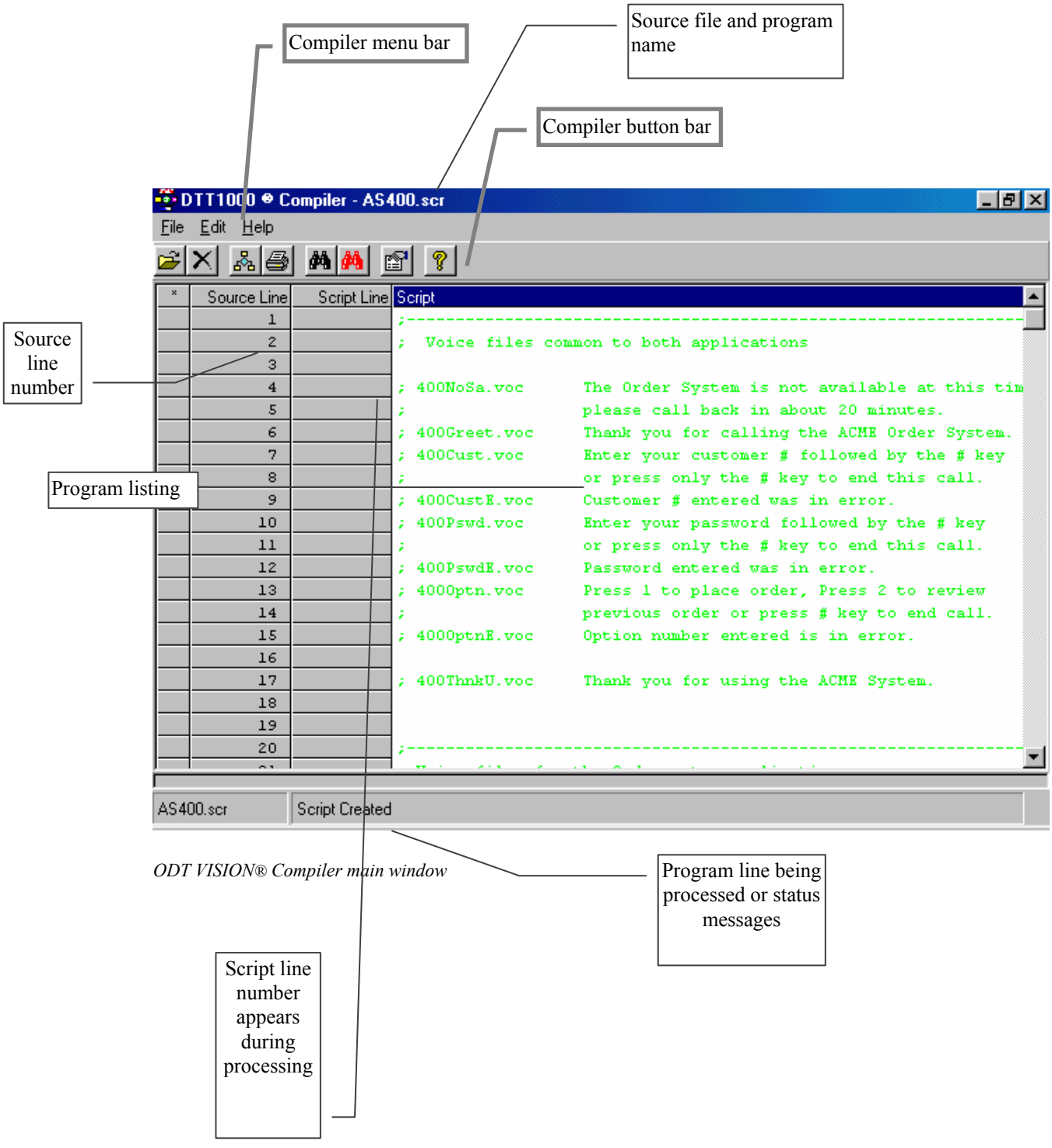
Option	Menu Hot Keys	Icon	
Open script file	Alt-F, O		Opens a file selection dialog window to select the script to compile.
Compile the script	Alt-F, C		Compiles the selected script.
Print the script	Alt-F, P		Prints a script that has already been compiled.
Remove a script	Alt-F, R		Removes a previously compiled script from the **ODT VISION®** system database.
Find	Alt-E, F		Opens a find dialog that allows you to enter a search string.
Find errors	Alt-E, E		Finds errors in a compiled script.
Properties	Alt-E, P		Allows changes to Compiler program options.
Exit	Alt-F, X		Exits the Compiler program.

The **ODT VISION®** introduction screen is displayed, followed by the Compiler main window.

If this is the first time you have started a **ODT VISION®** program since the software was installed, you will be prompted for your Company or User name.

ODT VISION® Compiler main window

Below is an example of the **ODT VISION®** Compiler main window with the various components of the window identified.



ODT VISION® Compiler main window

Compiling a script

Opening a script source file

To create a **ODT VISION®** compiled script, you must first open the source file that contains the application script. The default extension for script source files is .ODT although any file extension can be opened.

Starting the compile

When you select the compile option, a window is displayed to allow you to name the compiled script. The name of the source file is displayed as the default compiled script name. Click the OK button to start the script compile.

NOTE: It is recommended that the name of the source file is included in the compiled script name. This will help you remember the source name if changes need to be made at a later time.

Normal and abnormal compiles

When a compile has completed, the results of the compile are displayed at the bottom of the Compiler main window.

If the compiled script had any syntax errors, they are displayed in the listing, usually under the line that has the error. Error lines are identified in the listing as a red line with an "E" in the line number column. A message number will be displayed between the [ ] in the error message line. See the section on **ODT VISION®** Compiler messages for a list of compiler errors.

If the compiled script has errors, repeat the process of editing and compiling until all errors have been removed from the script.

NOTE: Do not attempt to run a compiled script that has not had a normal compile.

If the script compiled normally, test the compiled script using the **ODT VISION®** Monitor program..

Printing a compiled listing

After a compile, you can print a listing of the script to your default Windows printer. Select the option for printing a script from the menu or the toolbar. Printing options can be set in the Compiler properties window.

Chapter 3 - Script Language Reference

Overview

This section introduces you to the basic elements of the **ODT VISION®** scripting language, including Variables, Constants, and Commands. We will also cover the conventions used in this manual and the syntactical rules of the script language itself.

What is a script?

A script file is an ASCII text file that you create that contains commands to be executed by the **ODT VISION®** Operator program. These commands control how **ODT VISION®** will interact with the user that calls into the system. The ASCII text file can be created using any text editor available such as Windows Notepad. The Compiler program is then used to convert the script into a machine readable version that is placed in the ODT VISION.MDB database.

Each line in a script begins with:

- A Command, usually followed by one or more parameters, which control the way the Command functions.
  or
- A variable that will be modified by an Command; for example, **Extension = Price * Quantity.** In this example, the variable **Extension** will be modified from the result of the multiplication Command of **Price** times **Quantity**.
  or
- A line label (in the form of **LABEL:)** that designates the target location for a **GoTo** or **GoSub** Operation.
  or
- A comment line (identified with a semicolon (;) at the start of the line. This allows you to place notes in your script that will be ignored by the Compiler program.

What is a Control Window script?

The Control Window is a special operator that can run Commands when the **ODT VISION** system starts, or at certain times. This allows you to perform startup and time related functions such as updating databases, uploading or downloading batch files, shutting down the system at certain times, and many other maintenance related functions. A Control Window script is created like any other **ODT VISION** script except that phone related Commands can not be used.

The following Commands are valid only in a Control Window script.

OnTime

OnTimeEnd

LineStart

LineStop

LineStatus

RestartSystem

Rules for Control Window scripts:

- Up to ten (10) **OnTime** Commands can be performed in a Control Window script.
- Make sure that **OnTime** commands do not overlap. If a 1:00am **OnTime** Command is executing and it takes more than an hour to complete, a 2:00am **OnTime** Command would NOT execute.
- Control Window scripts must start with any **OnTime** commands followed by a **OnTimeEnd** Command.
- Only 1 **OnTimeEnd** Command per **OnTime** Command allowed.
- The Control Window script must complete with an **End** command before any **OnTime** Commands will execute.
- Put **Wait** Commands in your Control Window script to keep it from "consuming" your **ODT VISION** system.
- No “Phone” Commands can be performed in a control window script.

Script language elements

Commands

Commands in your script control the **ODT VISION®** Operator program and are the building blocks of your telephony application. A Command may have one or more parameters that further define the way the Command works. Commands and their associated parameters have the following characteristics:

- Each Command must be on a separate line in the script.
- The maximum line length can be up to 32000 characters (may be limited by system memory or the editor that is used).
- Commands and the associated parameters are separated by a space or a comma depending on the Command.
- Commands and parameters are not case sensitive (except for string information that is contained between two quotes; as in "This is a string constant").

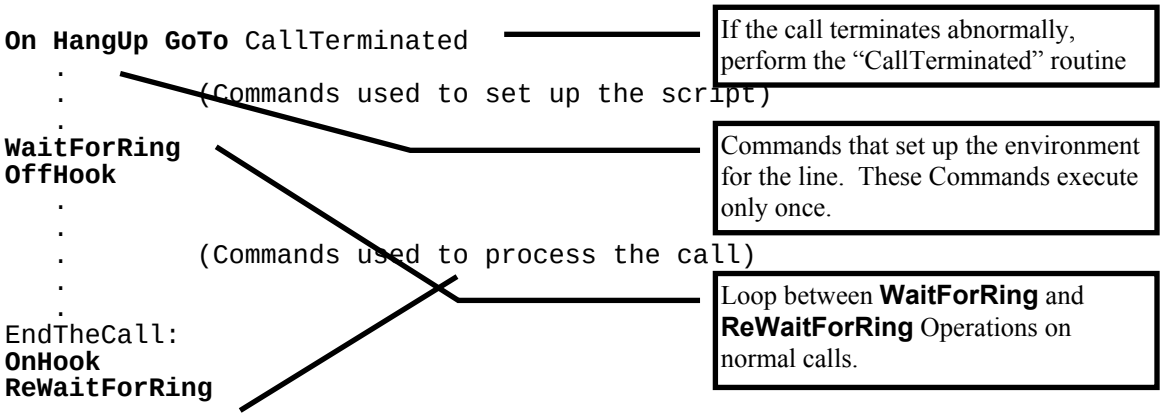
Script flow

The script will start executing with the first non-comment line and will continue until the **WaitForRing** or **End** Command is encountered.

If **WaitForRing** is encountered, execution will halt at this statement until a call comes in, then the script will continue processing at the first statement following the **WaitForRing**. When the caller hangs up, the script will branch to the label identified in the **ON HangUp** Command (if used), or will continue processing until the **ReWaitForRing** or **End** Commands are encountered.

If the **ReWaitForRing** Command is encountered, the script will branch back to the **WaitForRing** line and wait for the next call. If the **End** Operation is encountered, call processing for the line will stop.

The following example shows the basic script flow of an inbound call script…..




```
CallTerminated:
MsgBox "Call Ended Abnormally"
GoTo EndTheCall
```

Data Types

The ODT VISION® system supports the following data types:

- Numeric (numbers both positive and negative)
- String (a group of alphanumeric characters)
- Date/Time (a special data type that can contain a complete date and/or time)

Variables

ODT VISION® supports named elements called variables that are used to store values used within a script. Variables can then be used as parameters to Commands, or can used as a result or an operand in an expression. All variables used in a ODT VISION® script are global and can be defined or used anywhere in the script. There are three types of variables used by the ODT VISION® system. A description of each variable type is listed below:

User-defined Variables

User-defined variables are created by you, for use in your scripts. They are defined automatically by using the variable anywhere in the script. User-defined variables must follow the naming conventions described in the section: "Naming items in a script". It is recommended that the names that you create for variables be as descriptive as possible for maximum readability of your script. A user-defined variable can be of the type numeric, string, or date/time and can change type dynamically depending on the Command that created the value in the variable. For example, if the following operation was used in a script: **TotalDollars = Units * Price**, the variable TotalDollars would adopt a numeric data type because it was a result of a numeric calculation. See the section on variable type conversions for more information.

File Variables

File variables are variables that were pre-defined in a Database Table, that was created to be used by your application. These file variables are sometimes referred to as "fields" or "columns" by other languages and systems. File variables have a data type that was defined at the time that the Table was created. Since file variables have a pre-defined data type, any change to the data in the variable will be made compatible with the data type of the file variable automatically. For example, if a file variable call **ZipCode** was defined in a Table as a data type of string, and a numeric value was put into the field, the value would be converted to a string automatically. See the section on variable type conversions for more information.

File Variables are specified in a script as follows:

TableName.FieldName

The Name of the table as defined in the **Open** Command, followed by a period (.), followed by the name of the Field or Column.

If TableName or FieldName contains spaces or other special characters, then brackets must be placed around the names to make them valid in a script. For example:

[Customer Master].[Customer Name]

System Variables

System variables are pre-defined variables that can be used to gather information about the system, or can be used to control the way the system operates. Some system variables are 'read-only' and the values that are in these variables are created by the system automatically. Even though you cannot change these variables, they can be used throughout your script to get information about the system itself. Other system variables can be modified and have a data type that is pre-defined by the system. Any changes to the value in a system variable will be made compatible with the variable data type automatically. See the section on variable conversions for more information.

Variable conversions

If the result of a Command is user-defined variable, then the variable will adopt the data type that is returned from the Command. For example, if a numeric Operation such as
Total = Price * Quantity is performed, the result variable (**Total**) will be changed to a numeric data type.

If the result returned from a Command is a File or System Variable, then the value returned from the Command will be converted to a variable type compatible with the result variable.

Variables can be converted from one type to another, although this is not usually necessary because the ODT VISION® system makes any conversions required automatically when the wrong data type is passed as a parameter or used in an expression. The following is a list of Commands you can use to perform conversions from one data type to another:

Operation	Convert From	Convert To
Value	String	Numeric
String	Numeric	String
Format	Numeric	String
Format	Date/Time	String
ASCII	String	ASCII value
CHAR	ASCII value	String

When a string is converted to a number, the string is interpreted one character at a time starting at the left side of the string. Spaces at the start of the string will be ignored when evaluating the number. The number will be processed left to right until the first non-numeric character is processed. For example, the string
“ 123.45AB7890”
will be returned as the number **123.45**. The valid characters in a number are 0 through 9, a period (for a decimal point), and a negative or plus sign (-,+).

When a number is converted to a string, the first character will be a negative sign (-) if the number is negative. No plus (+) sign is supplied if the number is positive. A decimal point will only be included if the number has a fractional part. Listed below are some sample conversions from numbers to strings:

Convert from number:	To string:
1234.56	“1234.56”
-123.4	“-123.4”
-3342	“-3342”

Constants

Constants are similar to variables in that you define them yourself in the script. Constants are used when the value needed in the script does not change, such as a parameter that is passed to a Command. Constants can be numeric, string, or a date/time data type.

Numeric constants are just keyed as a number in a parameter or an expression. If a parameter starts with any of the following characters, it is considered a numeric constant:
-, +, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or a period (.)
In the example, **Extension = Total * .85** the **.85** is a numeric constant.

String and Date/Time constants must be surrounded by double quotes; for example,

```
FullName = "Jack" BCAT "Sprat".
If TimeNow > "23:00pm" Then
      .
      .
Endif
```

Labels (tags)

Labels are marked places in a script that control can be transferred to by a **GoTo** or **GoSub** Operation. A label must be on a line all by itself and must be a valid name followed by a colon (:). Labels must follow the naming conventions described in the section: "Naming items in a script". It is recommended that the names that you create for labels be as descriptive as possible for maximum readability of your script.

Naming items in a script

Variables, Commands, and labels all follow the same basic naming conventions:

- Names have a maximum of 30 characters
- Names begin with a letter, with each additional character in the name being a letter, number, or one of the following special characters: _, -, !, ?, $
- All pre-defined names (such as Commands or System Variables) are considered reserved words and cannot be used as a user-defined variable or label name.

NOTE:
If Table names, Field names, or Table Index names contains spaces or other special characters, then brackets can be placed around them to make them a valid name for use in a ODT VISION® script.
For example:
[Customer Master].[Customer Name]

Database elements

Database Container

The ODT VISION® system requires a file called ODT VISION.MDB that is used to store compiled scripts and to provide or store information for your ODT VISION® applications. This file, which is located in the directory where your ODT VISION® system was installed, is referred to as a Container because it stores all of the Tables (both internal Tables and external Database Table references) required by the ODT VISION® system. This file is a Microsoft Access (version 2000) compatible file that can be maintained using Microsoft Access (Version 2000 or higher).

The ODT VISION.MDB container stores the following information:

- Any Tables used by your ODT VISION® application.
- Any references to external Tables used by your application (referred to as 'Attached Tables')
- Compiled ODT VISION® scripts

Database file (Table)

One type of item that can be stored in the ODT VISION.MDB Container are the Tables that are used by your ODT VISION® applications. A Table is a collection of information about a particular subject. For example, one Table might contain information about products that you sell, while another Table might contain information about your customers.

All ODT VISION® applications use the same ODT VISION.MDB Container, but each application may access different Tables in the Container if needed. For example, if the application running on line 1 gives pricing information, it will probably have to reference an Inventory Table to validate the item exists and to look up the price of the items. If line 2 takes orders from your customers, it would probably reference a Customer Table to validate the customer number entered by the user, and the same Inventory Table used by line 1 for validation and price lookup. Both the Inventory and the Customer Tables would be created in, or attached to, the ODT VISION.MDB container..

Information in a Table can be maintained by one or all of the following methods:

- By using Microsoft Access (version 2000 or higher)
  ...and adding, changing, or deleting records from the Table
  ...or by importing records from another system or program
  ...or by attaching to an external data source such as a DBASE file
- By using any other Microsoft Access compatible program that can perform record level maintenance to Tables in an Access Database container.

Records

A record is a collection of information about one item in a Table. Some database systems refer to this as a Row. For example, in a Customer Table, a record would be the information that is kept on one customer. Each record in the Table contains the same type of information for each customer.

For example, a customer record for a ODT VISION® application might contain:

- Customer Number
- Customer Name
- Password
- Discount %

Database Field

Each piece of this information that is kept in a record is referred to as a field. Some databse systems refer to this as a column. In the example above, there would be four fields in each customer record.

Indexes

Index definitions are created for a Table as a way to view your Table records in a specific order. This makes finding records in a large Table faster and easier. It also it allows ODT VISION® applications to randomly access specific records in a Table without searching through all of the records. Indexes over a Table are not necessary for a ODT VISION® application to use the Table.

In the example Customer Table above, we could create an Index over the Customer Number field if we needed to validate customers in a ODT VISION® application. Having this Index would also allow you to quickly locate a specific customer in a ODT VISION® application that validates customers.

To recap:

- A ODT VISION.MDB Container stores all information about Tables, Attached Tables, and compiled ODT VISION® applications.
- A Table contains records about a specific type of information that needs stored for, or referenced by your ODT VISION® application.
- A record contains is a group of related information in a Table.
- A field contains one piece of data for one record in a Table.

NOTE:
If an Index name contains spaces or other special characters, then brackets can be placed around the name to make it valid for a ODT VISION® script. For example:

[Customer Number Index]

Flat file

Flat files are standard DOS ASCII files that are created outside of the ODT VISION.MDB Container. Flat files can be read or written by a ODT VISION® script or imported/exported from the Container by the Microsoft Access program. Flat files are sometimes necessary when data collected, or required by your script, needs to be used by another PC, Mini, or Mainframe computer.

Default Directories

Every Command that uses files has a default directory associated with the Command. The ODT VISION® Monitor program allows you to set the default Voice file directory. This allows you to use different Voice files for each line, even thought the lines may be using the same script. An example of how this might be used would be for multi-language support on different lines. The default directory can always be overridden by specifying a full or partial path in front of the file name supplied to the Command.

For the examples below, assume the following were true:

Current Drive when Windows was started: "C:"
Current Directory on C: "**C:\Program Files\ODT VISION"**
Default Directory for the Command being performed: **"C:\Program Files\ODT VISION\Voice Files"**

Then the following would be true:

DOSFile passed to the Command	DOS path/file used by the Command.
C:\DIR\FILE.EXT	C:\DIR\FILE.EXT
(Full absolute path was specified, no change will be made)	
C:\FILE.EXT	C:\FILE.EXT
(Full absolute path was specified, no change will be made)	
\DIR\FILE.EXT	**C:\Program Files\ODT VISION\Voice Files**\DIR\FILE.EXT
(Relative path was specified, so append it to the default directory for the Command)	
FILE.EXT	**C:\Program Files\ODT VISION\Voice Files**\FILE.EXT
(No path was specified, so prefix the file name with the default directory for the Command)	

DOSFile can not contain any (.), or (..) directory names. For example **'..\DATA\FILE.EXT'** would be invalid.

This page left intentionally blank

Script Commands

Developer manual conventions

The following conventions are used in this manual to describe the Commands and parameters used in the script language:

[ ]	Any parameter enclosed in square brackets is optional.
{ }	Curly brackets indicate that two or more parameters are required when multiple choices are available.
|	If multiple parameters are separated by a vertical bar, chose a single parameter from the list of choices.
...	Multiple parameters can be specified if the ellipse characters are displayed.
lowercase italic	Substitute a variable, expression, or a constant for a parameter shown in lowercase italic.
variable	A variable of any type that is returned from, or supplied to a Command.
numvar	A numeric variable that is returned from, or supplied to a Command.
strvar	A string variable that is returned from, or supplied to a Command.
dtvar	A date/time variable that is returned from, or supplied to a Command.
dvar	A date variable that is returned from, or supplied to a Command.
tvar	A time variable that is returned from, or supplied to a Command.
value	A value of any type that is supplied to a Command.
numval	A numeric value that is supplied to a Command.
strval	A string value that is supplied to a Command.
dtval	A date/time value that is supplied to a Command.
dval	A date value that is supplied to a Command.
tval	A time value that is supplied to a Command.
format	A string value that contains a formatting string that is supplied to a **Format, SpeakDate,** or **SpeakTime** Command.

NOTE: Script examples in this manual are presented in both upper and lower case. You may use either upper or lower case (or both) in your scripts to improve the readability of the script. No elements in a script are case sensitive except for data within quoted strings. For example, **WaitForRing, WAITFORRING,** and **waitforring** will all execute the same Command.

This page left intentionally blank

Comment lines

Description: Lines, or parts of lines that are comments.

Syntax:

; *comment*

Where:

comment Documentation placed in the script.

Notes:

Everything after a semicolon (;) that is not within two double quotes (") is considered a comment. The semicolon can be the first character on a line in which case the whole line is a comment, or it can follow a valid Command and parameters.

Example:

```
; This entire line is a comment line.
Name = "John"                    ; This is a comment (after the semicolon)
```

Assigning variables

Description: Assigns a variable the value of another variable or constant.

Syntax:

variable **=** *value*

Where:

value	Numeric, string or date/time value to assign.
variable	The result variable that will be assigned the numeric, string, or date/time value contained in the *value* parameter.

Notes:

If *value* is a user-defined variable that has not been assigned a value, then a blank (empty) string value ("") will be assigned.

The result of the assignment can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then it will adopt the data type of the *value* parameter. If the result variable is a file variable or a read/write system variable, then the value will be converted to a variable type compatible with the result variable type.

See the section on variables for more information on variable conversions.

Example:

```
RingCount = 1                ; RingCount will be a numeric data type.
Name = "John"                ; Name will be a string data type.
StartCount = RingCount       ; StartCount will be a numeric data type.
DateNow = "12/31/99"         ; DateNow will be a date data type.
```

Mathematical Commands

Description: Assigns a variable the result of a mathematical operation between two values.

Syntax:

numvar = numval1 **+** *numval2*	Add *numval2* to *numval1*.
numvar = numval1 **-** *numval2*	Subtract *numval2* from *numval1*.
numvar = numval1 ***** *numval2*	Multiply *numval1* by *numval2*.
numvar = numval1 **/** *numval2*	Divide *numval1* by *numval2* (return a real number).
numvar = numval1 **** *numval2*	Divide *numval1* by *numval2* (return a whole number).
numvar = numval1 **Remainder** *numval2*	Divide *numval1* by *numval2* (return the remainder).
numvar = numval1 **^** *numval2*	Raise *numval1* to the power of *numval2*.

Where:

numval1	Numeric value.
numval2	Numeric value.
numvar	The result variable that will be assigned the value calculated by the mathematical operation.

Notes:

Both *numval1* and *numval2* are converted to a numeric value before the mathematical operation is performed.

Numval2 must not contain a value of 0 for the divide operations (**/**, ****, **Remainder)** or an error will occur (Divide by 0 error) and 0 will be returned.

The result of the mathematical operation can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a number. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
TotalValue = 1                ; TotalValue is 1
TotalValue = TotalValue + 4   ; TotalValue is 5
TotalValue = TotalValue * 2   ; TotalValue is 10
TotalValue = 10 / 4           ; TotalValue is 2.5
TotalValue = 10 \ 4           ; TotalValue is 2
TotalValue = 10 Remainder 4   ; TotalValue is 2
TotalValue = 2 ^ 8            ; TotalValue is 256
TotalValue = "10" + 5         ; TotalValue is 15
```

Numeric Commands

Description: Assigns a variable the result of a numeric operation performed on a value.

Syntax:

numvar = **SquareRoot** *numval*	Return the square root of *numval*.
numvar = **Absolute** *numval*	Return the absolute value of *numval*.
numvar = **Int** *numval*	Return the integer part of *numval*.
numvar = **Int2** *numval*	Return the integer part of *numval*.

Where:

numval	Numeric value.
numvar	The result variable that will be assigned the value calculated by the numeric operation.

Notes:

Numval is converted to a numeric value before the numeric Command is performed.

Both the **Int** and **Int2** Commands return the integer portion (whole number) of *numval*. The only difference is the way that each Command handles negative numbers. **Int** returns the next larger negative number, while **Int2** returns the next smaller negative number.

The result of the numeric Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a number. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
TotalValue = SquareRoot 4      ; TotalValue is 2
TotalValue = Absolute -2       ; TotalValue is 2
TotalValue = Absolute  2       ; TotalValue is 2
TotalValue = Int2 -2.4         ; TotalValue is -2
TotalValue = Int2 2.4          ; TotalValue is 2
TotalValue = Int -2.4          ; TotalValue is -3
TotalValue = Int 2.4           ; TotalValue is 2
```

String concatenation Commands

Description: Assigns a variable the result of a concatenation of two values.

Syntax:

strvar = strval1 **&** *strval2*	Append *strval2* to *strval1*.
strvar = strval1 **Cat** *strval2*	Same as "&".
strvar = strval1 **BCat** *strval2*	Truncate *strval1* of any trailing blanks and append *strval2* to it with a single space in between.
strvar = strval1 **TCat** *strval2*	Truncate *strval1* of any trailing blanks and append *strval2* to it.
strvar = strval1 **DCat** *strval2*	Add a "\" to the end of *strval1* (if it is needed) to create a valid directory name and append *strval2* to it.

Where:

strval1	String value.
strval2	String value.
strvar	The result variable that will be assigned the string value created by the concatenation Command performed on *strval1* and *strval2*.

Notes:

Both *strval1* and *strval2* are converted to a string value before the concatenation Command is performed.

The result of the concatenation Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a string. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
FirstName = "JOHN     "
LastName = "SMITH"
DirName = "C:\DATA"
Name = FirstName & LastName          ; Name is "JOHN     SMITH"
Name = FirstName Cat LastName        ; Name is "JOHN     SMITH"
Name = FirstName BCat LastName       ; Name is "JOHN SMITH"
Name = FirstName TCat LastName       ; Name is "JOHNSMITH"
FileName = DirName DCat "File.ext"   ; FileName is "C:\DATA\File.ext"
```

Substring Commands

Description: Assigns a variable the result of the substring Command performed on a string value.

Syntax:

strvar = **Left** *strval*, *len*	Return the left side of *strval* for a length of *len.*
strvar = **Right** *strval, len*	Return the right side of *strval* for a length of *len*.
strvar = **Middle** *strval, start*	Return the right side of *strval* beginning at *start*.
strvar = **Middle** *strval, start, len*	Return the middle of *strval* for a beginning at *start* for a length of *len*.
strvar = **LeftTrim** *strval*	Return *strval* with all spaces removed from the left.
strvar = **RightTrim** *strval*	Return *strval* with all spaces removed from the right.
strvar = **Trim** *strval*	Return *strval* with all spaces removed from both sides.
numvar = **Split** *strval, [delim]*	Splits *strval* into separate string values wherever a blank is encountered (if *delim* is not specified) or wherever the *delim* character is encountered. Returns the number of separate string values that were created.
strvar = **Part** *partno*	Returns the part of the string that was created with the **Split** Command specified by the *partno* parameter. The *partno* paramter can be 1 to the number returned from the Split Command.

Where:

strval	String value that the substring Command will be performed on.
len	Numeric value of the length of the substring Command.
start	Numeric value of the starting position of the substring Command.
delim	A single character that specifies the delimiter that the **Split** Command will break the supplied string by.
partno	Numeric value that specifies the part of the string that will be returned from the **Split** Command.
strvar	The result variable that will be assigned the string created by the substring Command.
numvar	The result variable that will be assigned the numeric value created by the **Split** Command.

Notes:

Before the substring Command is performed, *strval* and *delim* will be converted to a string value, *len, start* and *partno* will be converted to a numeric value.

Len and *Start* should be a number between 0 and the length of *strval.*

Delim should be a single character that will be used to split the supplies string.

Partno should be a number between 1 and the length returned from the last **Split** Command.

The **Split** Command can be used to break sentences into words, or to separate values in a comma or tab delimited string.

The result of the substring Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a string. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
Name = "JOHN PAUL SMITH"
FirstName = Left Name, 4              ; FirstName is "JOHN"
LastName = Right Name, 5              ; LastName is "SMITH"
MidLastName = Middle Name, 6          ; MidLastName is "PAUL SMITH"
MidName = Middle Name, 6, 4           ; MidName is "PAUL"
NameParts = Split Name                ; NameParts is 3
FirstName = Part 1                    ; FirstName is "JOHN"
LastName = Part 3                     ; LastName is "SMITH"


Name = "           JIM          "
RightSide = RightTrim Name            ; RightSide is "JIM          "
LeftSide = LeftTrim Name              ; LeftSide is "          JIM"
MiddlePart = Trim Name                ; MiddlePart is "JIM"
```

Alpha Conversion Commands

Description: Assigns a variable the result of an Alpha Conversion Command performed on a numeric string.

Syntax:

strvar = **AlphaConvert1** *strval*	Returns a Alpha conversion of *strval*.
strvar = **AlphaConvert2** *strval*	Returns a Alpha conversion of *strval*.
strvar = **AlphaConvert3** *strval*	Returns a Alpha conversion of *strval*.
strvar = **AlphaConvert4** *strval*	Returns a Alpha conversion of *strval*.

Where:

strval	String value to convert.
strvar	The result variable that will be assigned the string value created by the conversion Command.

Notes:

The *strval* parameter will be converted to a string value before the Alpha conversion Command is performed.

The Alpha Conversion Commands are used to convert a string of numbers keyed by the user during a phone operation to a valid Alpha sequence. Using these Commands, your script can accept alpha and special character information from the numeric keypad of a touch-tone phone.

There are two methods of accepting alpha and special character input from a numeric phone:

The first method is called the Prefix method. Using the Prefix method, the user must enter a "*****" before each alpha or special character, followed by the letter or character on the keypad, followed by the position of the letter or character on the keypad. In other words, it takes 3 characters to specify a letter or special character. If an "*****" is not keyed first, the numbers on the keypad are assumed.

For example, if the user keyed **"*21*23*32789*13"**,
the Prefix method would return **"ACE789@".**

The Prefix method is accomplished using the **AlphaConvert1** and **AlphaConvert3** Commands.

The second method is called the Shift IN/Out method. Using the Shift IN/OUT method, the user must enter a "*****" to shift into alpha/special character mode. After Shifting IN, numbers are keyed in pairs with the first number representing the letter or character on the keypad, followed by the position of the letter or character on the keypad. This can be repeated until all alpha or special characters are keyed, then a "*****" is keyed again to Shift OUT of alpha/special character mode. In other words, it takes 2 characters to specify a letter or special character while Shifted IN. If Shifted OUT, the numbers on the keypad are assumed.

For example, if the user keyed **"*212332*789*13*"**,
the Shift IN/OUT method would return **"ACE789@".**

The Shift IN/OUT method is accomplished using the **AlphaConvert2** and **AlphaConvert4** Commands.

The **AlphaConvert3** and **AlphaConvert4** Commands are the proper way to process alpha input from any modern phone. **AlphaConvert1** and **AlphaConvert2** are retained for compatibility with previous versions of the ODT VISION system. The main difference is that the newer phones have Q and Z actually on the keypad so **AlphaConvert3** and **AlphaConvert4** are used with these newer phones. Also, **AlphaConvert3** and **AlphaConvert4** have more special characters that can be keyed on the phone.

AlphaConvert1 and AlphaConvert2 keyboard

1	**[sp] . @ 1**	**2**	**ABC2**	**3**	**DEF3**
4	**GHI4**	**5**	**JKL5**	**6**	**MNO6**
7	**PRS7**	**8**	**TUV8**	**9**	**WXY9**
*****	**[reserved]**	**0**	**QZ*0**	**#**	**[reserved]**

AlphaConvert3 and AlphaConvert4 keyboard

1	**[sp] . @ 1 ? ! , _ & :**	**2**	**ABC2**	**3**	**DEF3**
4	**GHI4**	**5**	**JLK5**	**6**	**MNO6**
7	**PQRS7**	**8**	**TUV8**	**9**	**WXYZ9**
*****		**0**	*** # - 0 + / = $ % **	**#**	**[reserved]**

Example:

```
Flight:

      ClearDigits
      ;plays voice file to enter flight number
      ReturnCode = Play "Aflight.vox",0,"@"
      Returncode = GetDigits 15,"#",25
      Flightnumber = DigitBuffer
      Flightnumber = Alphaconvert4 FlightNumber
```

Number Conversion Commands

Description: Assigns a variable the result of a Number Conversion Command performed on an Alphanumeric string.

Syntax:

strvar = **NumberConvert1** *strval* Returns a Number conversion of *strval*.
strvar = **NumberConvert2** *strval* Returns a Number conversion of *strval*.

Where:

strval String value to convert.
strvar The result variable that will be assigned the string value created by the conversion Command.

Notes:

The *strval* parameter will be converted to a string value before the Number Conversion Command is performed.

The Number Conversion Commands are used to convert a string of alphanumeric characters into a valid phone numeric keying sequence. For example, by using these Commands, your script could convert database fields into valid keying sequences so that the user does not have to resort to keying alphanumeric information using the phone keypad.

There are two methods of converting alphanumeric information into a numeric string. Both methods are identical other than how they handle the characters Q and Z. The **NumberConvert2** Command is the proper way to process alpha data for any modern phone. **NumberConvert1** is retained for compatibility with previous versions of the ODT VISION system.

Both commands drop any characters that are not letters or numbers. Numbers are not changed in the string. Since special characters are dropped from the supplied string during the conversion, the string value returned can be shorter than the input string.

NumberConvert1

Character	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Returns	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	0	7	7	8	8	8	9	9	9	0

NumberConvert2

Character	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Returns	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	7	8	8	8	9	9	9	9

Example:

```
; Convert part numbers from a database to a valid keying sequence
; and update this value back to the database.

; In this example, if DB.PartNo was “ADG-475%2”, DB.NumPartNo would
; contain “2344752” after the conversion.

DB.NumPartNo = NumberConvert2 DB.PartNo
```

Formatting Commands

Description: Assigns a variable the result of a formatting Command performed on a numeric, string, or date/time value.

Syntax:

strvar = **Format** *value, format* Returns *value* as a string formatted as defined by *format.*

Where:

Value	Numeric, string, or date/time value to format.
format	A string of characters that specifies the format of the *value* parameter that will be returned. A predefined 'named' format name can also be used.
Strvar	The result variable that will be assigned the formatted string of *value.*

Notes:

The *format* parameter will be converted to a string value before the **Format** Command is performed.

When formatting numbers, *format* can contain up to three formatting strings, separated by a semi-colon (;). If only one format string is specified, it will be used for positive, negative, and zero numbers. If two format strings are used, the first formatting string will be used for positive and zero numbers, and the second string will be used for negative numbers. If all three formatting strings are specified, the first will be used if the number is positive, the second if negative, and the third if zero. In the first example below, two formatting strings are specified.

See the section on formatting strings for more information on the characters that are available to use for formatting.

The result of the **Format** Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a string. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
Qty1 = 5.01 * -2
x =  Format Qty1, " ####0.00;-####0.00"          ; Returns '-10.02'

TodaysDate = DATETIME
td = Format TodaysDate, "DDD, MMM, D, YYYY"
; The line above would return a string representation of the date value
; in TodaysDate.  An example would be 'Wed, Dec 12, 1994'
```

Formatting (Case) Commands

Description: Assigns a variable the result of a formatting Case Command performed on string value.

Syntax:

strvar = **UpperCase** *strval*	Returns *strval* as an all uppercase string.
strvar = **LowerCase** *strval*	Returns *strval* as an all lowercase string.
strvar = **NameCase** *strval*	Returns *strval* as a proper name case string.
strvar = **SentenceCase** *strval*	Returns *strval* as an proper sentence case string.

Where:

Strval	String value that will be converted.
strvar	The result variable that will be assigned the formatted string.

Notes:

The **NameCase** Command will set the first character of each word to upper case. All other letters will be returned as lower case.

The **SentenceCase** Command will set the first character of each sentence to upper case. All other letters will be returned as lower case.

The result of Case Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable data type will be converted to a string. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
Sentence = “NOW is the TIMe. LET US go.”
FullName = “JOHN Q. SMITH”

T = UpperCase Sentence        ; Returns “NOW IS THE TIME. LET US GO.”
T = LowerCase Sentence        ; Returns “now is the time. let us go.”
T = NameCase FullName         ; Returns “John Q. Smith”
T = SentenceCase Sentence     ; Returns “Now is the time. Let us go.”
```

Miscellaneous string Commands

Description: Assigns a variable the result of the string Command performed on other strings or assigns a variable the result of Commands that create strings.

Syntax:

numvar = **ASCII** *strval* — Return the ASCII code for the first character in the string *strval*.

strvar = **Char** *numval* — Return the character for the ASCII code in *numval*.

strvar = **String** *numval* — Return a string representation of the number in *numval*.

numvar = **Value** *strval* — Return the number that is contained in the string *strval*.

numvar = **Search** *start, searchin, searchfor*
Return the position of the string *searchfor* in the string *searchin* starting the search at character position *start*.

numvar = **Search** *searchin, searchfor*
Return the position of the string *searchfor* in the string *searchin* starting the search at the first character position of *searchin*.

numvar = **Length** *strval* — Return a number which is the length of string *strval*.

Where:

strval	String value.
numval	Numeric value.
start	Numeric value that specifies the starting position of a search.
searchin	String value to search.
searchfor	String value to search for.
numvar	The result variable that will be assigned the numeric value created by the Command.
strvar	The result variable that will be assigned a string value created by the Command.

Notes:

Strval, searchin, and *searchfor* are converted to a string value, *Numval* and *start* are converted to a numeric value before the Command is performed.

The **ASCII** Command returns the ASCII value of the first character of *strval*.

The **Char** Command returns a single ASCII character for the number in *numval. Numval* must be between 0 and 255 or an error will occur.

The **String** Command is similar in function to the **Format** Command for numeric values except a formatting string is not specified. *Numval* must be between -9999999999 and 9999999999.99999 (positive or negative).

The **Value** Command is used to convert a string to a number. *Strval* will be evaluated as a number, a character at a time until the first non-blank, non-numeric character is encountered. These Commands are not usually needed as conversions from one data type to another is handled automatically by the system.

The **Search** Command is used to determine the position of a string within another string. The starting position of the search is optional, and if not specified, the search will begin at position 1. If the search string is not found in the string to search, a value of 0 (zero) is returned from the **Search** Command.

The result of the String Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
Name = "JOHN PAUL SMITH"
x = ASCII "A"                          ; x is 65
x = Char 34                            ; x is "
x = String -34.2                       ; x is "-34.2"
x = String 34.2                        ; x is " 34.2"
x = Value "34.2"                       ; x is 34.2
x = Value "A34.2"                      ; x is 0
x = Search Name, "PAUL"                ; x is 6
x = Search 10, Name, "PAUL"            ; x is 0
x = Length Name                        ; x is 15
```

Date Commands

Description: Assigns a variable the result of a date and/or time Command.

Syntax:

dtvar = **StartDateTime**	Returns the date/time value of when the line went "Off-Hook".
dvar = **StartDate**	Returns the date value of when the line went "Off-Hook".
dtvar = **DateTime**	Returns the date/time value of the current date and time.
dvar = **Date** *[dtval]*	Returns the date value of the current date (if *dtval* is not supplied), or the date in *dtval* (if supplied).
numvar = **Month** *[dtval]*	Returns the month of the current date (if *dtval* is not supplied), or the month in *dtval* (if supplied). Returns a value of 1 to 12.
numvar = **Day** *[dtval]*	Returns the day of the current date (if *dtval* is not supplied), or the day in *dtval* (if supplied). Returns a value of 1 to 31.
numvar = **Year** *[dtval]*	Returns the year of the current date (if *dtval* is not supplied), or the year in *dtval* (if supplied).
numvar = **Weekday** *[dtval]*	Returns the day of the week of the current day (if *dtval* is not supplied), or the day of the week in *dtval* (if supplied). Returns a value of 1 to 7.

Where:

dtval	Date/Time value that contains a valid date and/or time.
dtvar	The result variable that will be assigned the date/time value created by the Command.
dvar	The result variable that will be assigned the date value created by the Command.
numvar	The result variable that will be assigned the numeric value created by the Command.

Notes:

See the section on date and time system variables for related information.

Dtval is optional, but if supplied, it is converted to a date/time value before the Command is performed.

The result of the date/time Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
HoldDate = "12/31/1992"
Sd = StartDateTime      ; sd will contain the start time of the call
mm = Month HoldDate     ; mm will be 12
mm = Month              ; mm will have the month of today's date
wd = Weekday            ; wd will have today's day number of (1-7)
```

Time Commands

Description: Assigns a variable the result of a date and/or time Command.

Syntax:

dtvar = **StartDateTime**	Returns the date/time value of when the line went "Off-Hook".
t*var* = **StartTime**	Returns the time value of when the line went "Off-Hook".
dtvar = **DateTime**	Returns the date/time value of the current date and time.
t*var* = **Time** *[dtval]*	Returns the time value of the current time (if *dtval* is not supplied), or the time in *dtval* (if supplied).
numvar = **Hour** *[dtval]*	Returns the hour of the current time (if *dtval* is not supplied), or the hour in *dtval* (if supplied). Returns a value of 0 to 23.
numvar = **Minute** *[dtval]*	Returns the minutes of the current time (if *dtval* is not supplied), or the minutes in *dtval* (if supplied). Returns a value of 0 to 59.
numvar = **Second** *[dtval]*	Returns the seconds of the current time (if *dtval* is not supplied), or the seconds in *dtval* (if supplied). Returns a value of 0 to 59.

Where:

dtval	Date/Time value that contains a valid date and/or time.
dtvar	The result variable that will be assigned the date/time value created by the Command.
tvar	The result variable that will be assigned the time value created by the Command.
numvar	The result variable that will be assigned the numeric value created by the Command.

Notes:

See the section on date and time system variables for related information.

Dtval is optional, but if supplied, it is converted to a date/time value before the Command is performed.

The result of the Date/Time Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
tm = Time                  ; tm will have today's time as a time value
dt = Format tm, "hh:mm" ; dt will have a string version of the date value in tm
                           ; (Like 23:45)
```

Date and Time Modifier Commands

Description: Assigns a variable the result of a date and/or time Modifier Command.

Syntax:

dvar = **CreateDate** *month, day, year* Returns the date value from *month, day,* and *year*.
tvar = **CreateTime** *hours, minutes, seconds* Returns the time value from *hours, minutes,* and *seconds*.

dtvar = **AddInterval** *dtval, interval, type* Returns the date/time value after adding an interval.
dtvar = **SubInterval** *dtval, interval, type* Returns the date/time value after subtracting an interval.

numvar = **ExtractPart** *dtval, type* Returns the part of a date/time value specified by *type*.
numvar = **Interval** *dtval1, dtval2, type* Returns the interval between two date/time values.

Where:

dtval, dtval1, dtval2	Date/Time value that contains a valid date and/or time.
month, day, year	A valid set of month (1-12), day (1-31), and year.
hours, minutes, seconds	A valid set of hours (0-23), minutes(0-59), and seconds(0-59).
interval	A numeric value that contains an interval of time.
type	One of the valid interval types listed in the table below.
dtvar	The result variable that will be assigned the date/time value created by the Command.
dvar	The result variable that will be assigned the date value created by the Command.
tvar	The result variable that will be assigned the time value created by the Command.
numvar	The result variable that will be assigned the numeric value created by the Command.

Notes:

See the section on date and time system variables for related information.

Dtval, dtval1, and *dtval2* are converted to a date and time value, *month, day, year, hours, minutes, seconds,* and *interval* are converted to a numeric value before the Command is performed.

Type must be one of the valid interval types listed in the table below:

Type	Interval
yyyy	Year
q	Quarter
m	Month
y	Day of Year
d	Day
w	Weekday
ww	Week
h	Hours
n	Minutes
s	Seconds

The **CreateDate** Command is used to create a valid date value from individual month, day, and year values.

The **CreateTime** Command is used to create a valid time value from individual hours, minutes, and seconds values.

The **AddInterval** Command is used to add an interval to a date and/or time value.

The **SubInterval** Command is used to subtract an interval from a date and/or time value.

The **ExtractPart** Command is used to extract only on value from a date and/or time value.

The **Interval** Command is used to determine the length of time between two date and/or time values.

The result of the date/time Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
christmasDate = CreateDate 12, 31, 2004
midnight = CreateTime 0, 0, 0

; Add 1 hour to the delivery time
deliverywindow = AddInterval deliverytime, 1, “h”

; Subtract 1 month from the Event date
alertdate = SubInterval eventdate, 1, “m”

; Return the year from the date
returnYear = ExtractPart returndate, “yyyy”

; Return the number of weeks since the return date
numberofweeks = Interval todaysdate, returndate, “ww”

; Return the elapsed minutes of the delivery
elapsed = Interval deliverytime, starttime, “n”
```

Holiday Commands

Description: Assigns a variable the result of a Holiday Command.

Syntax:

tfvar = **ClosedHoliday** *[dval]*	Returns *True* if today (if no date is supplied) is a holiday, or if the date supplied is a holiday.
strvar = **HolidayName** *[dval]*	Returns the name of today's holiday (if no date is specified), or the name of the holiday on the specified date.
numvar = **HolidayNumber** *[dval]*	Returns the number of today's holiday (if no date is specified), or the number of the holiday on the specified date.
dtvar = **HolidayDate** *holidayno [, year]*	Returns the date of the holiday number specified.

Where:

dval	Date value that contains a valid date.
holidayno	A valid holiday number (0-21) for standard holidays or (22-36) for custom holidays.
year	A valid year.
tfvar	The result variable that will be assigned the true/false value created by the Command.
strvar	The result variable that will be assigned the string value created by the Command.
dtvar	The result variable that will be assigned the date/time value created by the Command.
numvar	The result variable that will be assigned the numeric value created by the Command.

Notes:

See the section on date and time system variables for related information.

Dval is converted to a date and time value, *holidayno* and *year* are converted to a numeric value before the Command is performed.

The **ClosedHoliday** Command is used to check if today is a holiday (if no date is supplied as a parameter), or if the supplied date is a holiday. Returns true if today or the supplied date is a holiday. Only dates marked as holidays during setup are acknowledged as holidays by this Command.

The **HolidayName** Command returns the name of the today's holiday (if no date is supplied as a parameter), or the name of the holiday for the date supplied. Returns blanks if the date is not a holiday. If the date is a custom holiday that was entered during Holiday setup, a custom Holiday of "H1" to "H15" is returned. The holiday name returned is not necessarily a "closed" holiday, only the name of the holiday or custom holiday number. Use the **ClosedHoliday** Command to check if you are closed on a specific date. The name returned could be used to speak the holiday name in text-to-speech applications.

The **HolidayNumber** Command returns the number of today's holiday (if no date is supplied as a parameter), or the number of the holiday for the date supplied. Returns -1 if the date is not a holiday. If the date is a custom holiday that was entered during holiday setup, a customer holiday number of 22 to 36 is returned. The holiday number returned is not necessarily a "closed" holiday, only the number of the holiday or a custom holiday number. Use the **ClosedHoliday** Command to check if you are closed on a specific date. The number returned can be tested in the script to speak custom announcements based on holidays or special days.

The **HolidayDate** Command returns the date of a specific Holiday number for the current year (If no year is supplied as a parameter), or the date of a holiday for the number and year supplied. Valid Holiday numbers are 0 to 36. The holiday date returned is not necessarily a "closed" holiday, only the date of the holiday or custom holiday. Use the **ClosedHoliday** Command to check if you are closed on a specific date. A zero (0) date is returned if an invalid Holiday number is passed as a parameter,.

The result of the Holiday Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

See the Holiday Name and Number table at the end of this section for a table of Holiday.

Examples:

```
; ClosedHoliday Command example:
; Check to see if today is marked as a holiday
areclosed = ClosedHoliday
If areclosed = True then
      Speak "Closed.voc"
      Goto EndCall
Endif



; HolidayName Command example:
; Check today's date to see if it is a holiday.

chkHoliday = HolidayName

; HolidayName returned a blank so it is not a holiday.
If chkHoliday = "" then
      Goto PassHoliday

; Holiday "H1" is the "Closed for Inventory" day
Elseif chkHoliday = "H1" Then
      Play "CloseInv.VOC"
      Goto EndCall

; Today is a holiday, but not a custom holiday.
; Use text-to-speech to say the holiday.
Else
     VoiceText "Today is "
     VoiceText chkHoliday
     Rtn = VoiceSpeak
```

```
Endif

PassHoliday:


; HolidayNumber Command example
; Check today's date to see if it is a holiday

chkHoliday = HolidayNumber

; HolidayNumber returned a -1 so it is not a holiday.
If chkHoliday = -1 then
      Goto PassHoliday

; Custom holiday 1 (#22) is our "Closed for Inventory" day.
Elseif chkHoliday = 22 Then
      Play "CloseInv.VOC"
      Goto EndCall

; Today is a holiday, but not our custom holiday
Else
      ; Check to see if we are closed
      AreClosed = ClosedHoliday
      If AreClosed = True then

            ; We are closed today, so tell the caller the holiday.
            ; For example……
            ; "Today is"      (Voice file)
            ; "Christmas"     (TTS)
            ; "We are closed" (Voice file)
            varHoliday = HolidayName
            Speak "TodayIs.VOC"
            VoiceSpeak varHoliday
            Rtn = Speak "WeClosed.VOC"
            Goto EndCall
      Endif
Endif

PassHoliday:



; HolidayDate Command example:

; Check today's date to see if it is our Inventory date (Holiday # 22).

InventoryDate = HolidayDate 22

; If today is inventory date, tell them we are closed for inventory.
If Date = InventoryDate then
      rtn = Play "CloseInv.VOC"
      Goto EndCall
Endif
```

Holiday names and numbers

Holiday Number	Holiday Name
0	"New Year's Day"
1	"Martin Luther King Day"
2	"Groundhog Day"
3	"Valentine's Day"
4	“President's Day"
5	"Saint Patrick's Day"
6	"Tax Day"
7	"Easter"
8	"Mother's Day"
9	"Memorial Day"
10	"Flag Day"
11	"Father's Day"
12	“Independence Day"
13	"Labor Day"
14	"Columbus Day"
15	"Halloween"
16	"Election Day"
17	"Veterans' Day"
18	"Thanksgiving Day"
19	"Christmas Eve"
20	"Christmas"
21	"New Years Eve"
22-36	H1-H15 Custom Holidays

Open/Closed Commands

Description: Assigns a variable the result of an Open/Closed Command.

Syntax:

tfvar = **ClosedTime** *[tval]* Returns True if we are now closed (if no time is supplied), or if at the time supplied, we are closed.

tfvar = **Closed** *dtval* Returns True if we are now closed (if no date and time are supplied), or if at the date and time supplied, we are closed.

tvar = **DateOpenTime** *[dval]* Returns the Open time today (if no date is supplied), or the Open time on the date supplied.

tvar = **DateCloseTime** *[dval]* Returns the Close time today (if no date is supplied), or the Close time on the date supplied.

tvar = **DayOpenTime** *[dayofweek]* Returns the Open time today (if no *dayofweek* is supplied), or the Open time on the *dayofweek* supplied.

tvar = **DayCloseTime** *[dayofweek]* Returns the Close time today (if no *dayofweek* is supplied), or the Close time on the *dayofweek* supplied.

Where:

dtval	Date/Time value that contains a valid date and time.
tval	Time value that contains a valid time.
dval	Date value that contains a valid date.
dayofweek	A number from 1 to 7 representing the day of the week.
tfvar	The result variable that will be assigned the True/False value created by the Command.
dvar	The result variable that will be assigned the date value created by the Command.
tvar	The result variable that will be assigned the time value created by the Command.

Notes:

See the section on date and time system variables for related information.

Dtval is converted to a date and time value, *tval* is converted to a time value, *dval* is converted to a date value, and *dayofweel* is converted to a numeric value before the Command is performed.

The **ClosedTime** Command is used to check if you are now closed (if no time is supplied as a parameter), or if we are closed at the supplied time. Returns true if we are closed now or at the supplied time. Only checks the time to see if you are closed and does not take the date into account. Use this command when you have already checked to see if you are closed due to a holiday. To check both the date and time together, use the **Closed** Command.

The **Closed** Command is used to check if you are now closed (if no date and time are supplied as a parameter), or if we are closed at the supplied date and time. Returns true if we are closed now or at the supplied time. This Command checks both the date against the marked holidays, and the time against the Open/Closed times to see if you are closed. If a parameter is supplied to this Command, it must contain both a valid date and time value.

The **DateOpenTime** and **DateCloseTime** Command returns to Open or Closed time today (if no date is supplied as a parameter), or the Open or Closed time for the supplied date. Returns -1 if we are marked as closed on the requested date.

The **DayOpenTime** and **DayCloseTime** Command returns to Open or Closed time today (if no day of the week is supplied as a parameter), or the Open or Closed time for the supplied day of the week. The day of the week to check can be passed as a parameter to these commands where 1=Sunday through 7=Saturday. Returns -1 if a number less than 1 or greater then 7 is passed as a parameter.

The result of the Open/Closed Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will adopt the data type that is returned from the Command. If the result variable is a file variable or a read/write system variable, then the value will be converted to a data type compatible with the result variable.

See the section on variables for more information on variable conversions.

Examples:

```
; ClosedTime Command example:
; Check now to see if we are closed

areclosed = ClosedTime
If areclosed = True then
      Speak "WeClosed.VOC"
      Goto EndCall
Endif



; Closed Command example 1:

; Check to see if we are now closed.

areclosed = Closed
If areclosed = True then
      Speak "Closed.VOC"
      Goto EndCall
Endif



; Closed Command example 2:

; Check to see if we will be closed at the time the user entered.

chktime = dateSelected + timeSelected
areclosed = Closed chkTime

; If the user selects a date and time that we are closed,
; tell them, and have them enter another time.
If areclosed = True then
      Speak "NotOpen.VOC"
      Goto TryAgain
Endif

; If we made it here, we are open.
; Now, check to see if the appointment is open by
; checking an appointment database…….



; DateOpenTime/DateCloseTime Command example 1:

TimeWeOpen = DateOpenTime
TimeWeClose = DateCloseTime
If TimeEntered < TimeWeOpen Then
      Speak "TooEarly.VOC"
Elseif TimeEntered > TimeWeClose Then
      Speak "TooLate.VOC"
Else
      ; Here is where you might check a database to determine if the
      ; appointment time entered is open.
Endif
```

```
; DateOpenTime/DateCloseTime Command example 2:

TimeWeOpen=DateOpenTime
TimeWeClose=DateCloseTime

TC = TimeWeOpen

; Check every 10 minutes starting at our Open time to see
; if there is an appointment open

CheckTimeLoop:

      ; Check a database and see if there is an appointment scheduled.
      ; Check it against the TC variable which starts at our OPEN time.

      If AppointmentOpen = False Then

            ; Database said the appointment was not open so
            ; increment TC by 10 minutes and check the time again.

            TC = AddInterval TC, 10, "n"

             ; Check to see if the new time is after our close time.
             ; This would occur if all appointments are scheduled.
             If TC > TimeWeClose then
                  ; Tell the user all times are taken for the day.
                  Speak "NoTimes.VOC"
                        Goto GetAnotherDay
             Endif
             Goto CheckTimeLoop

      Else
            ; Tell them…….
            ; "An appointment is open at"
            ; "10:30 am"
            ; "Would you like the appointment?"
            Speak "OpenAppnt.VOC"
            SpeakTime TC, "hh:nn am/pm"
            Speak "AskOK.VOC"
      Endif


; DayOpenTime/DayCloseTime Command example:

; Tells the caller our open and close times for Sunday

TimeWeOpen=DayOpenTime 1
TimeWeClose=DayCloseTime 1

Speak "OpenFrom.VOC"
SpeakTime TimeWeOpen, "h:nn am/pm"
Speak "until.voc"
SpeakTime TimeWeClose, "h:nn am/pm"
Speak "onSunday.VOC"
```

Script flow Commands (If-Then-Else)

Description: Allows the conditional execution of statements based on an expression being evaluated as **True**.

Syntax:

If *expression* **Then** *operation* ; If *expression* is **True**, then perform *operation*.

If *expression* **Then** ; If *expression* is **True**, then perform all script lines up until the
: *;* next **ElseIf, Else**, or **EndIF**.
:

[**ElseIf** *expression* **Then**] ; If all prior **If** and **ElseIf** tests are **False**, and this **ElseIf** *expression* is
: ; **True**, then perform all script lines up until the next **ElseIf**,
: ; **Else**, or **EndIF**.
:

[**Else**] ; If all prior **If** and **ElseIf** tests are **False**, then perform all script lines
: **;** up until the **EndIF**.

EndIf

Where:

expression An expression that evaluates to **True** or **False**. Comparison operators for expressions can be **<** (less than), **>** (greater than), **<=** (less than or equal to), **>=** (greater than or equal to), **=** (equal), or **<>** (not equal).

operation The Command to perform if *expression* evaluates to **True**. *Operation* can be any valid script Command.

Notes:

If *expression* is a single field, then the field is tested for a **True** condition based on the following:

If field data type is:	Expression is **True** if field is
Numeric	not 0
Alpha	not a length of 0 ("")

If *expression* compares two fields or constants with a comparison operator, then the second field will be converted to the same data type, if necessary, before the comparison takes place.

See the chapter on variables for more information on variable conversions.

When two values are compared, the comparison operator can be one of the following:

Comparison Operator	Test for:
<	Less than
>	Greater than
<=	Less than or Equal
>=	Greater than or Equal
=	Equal
<>	Not Equal

Example:

```
If chkThis Then Goto DoIt            ; If chkThis is numeric and is not equal
                                     ; to 0, then go to DoIt.


If Counter < 2 Then
      MsgBox "Counter is less than 2"
ElseIf Counter = 2
      MsgBox "Counter is 2"
Else
      MsgBox "Counter greater than 2"
EndIf
```

Script flow Commands (If-Exists)

Description: Allows the conditional execution of statements based on the existence of a file.

Syntax:

If Exists *file* **Then** *operation* ; **If Exists** is an alternate form of the **If** Command used to test for the ; existence of a file. (See "If-Then-Else" for more information)

IF Exists *file* **Then**
:
[**ElseIf Exists** *file* **Then**]
:
[**Else**]
:
EndIf

Where:

File	String value that contains the file to check for on disk. *File* may contain a relative or absolute path along with the file name.
operation	The Command to perform if *file* exists. *Operation* can be any valid script Command.

Notes:

If Exists is an alternate version of the **If** Command, used to test for the existence of a file. (See the "**If-Then-Else**" Operation for more information)

Example:

```
; Both of the following examples test for the existence of "C:\temp\TestFile.dat"
; and deletes the file if it exists.

tf = "C:\temp\TestFile.dat"

If Exists "C:\temp\TestFile.dat" Then Kill "C:\temp\TestFile.dat"

If Exists tf Then
      Kill tf
Endif
```

Script flow Commands (Jumps and subroutines)

Description: Jump to a specific place in the script or execute a subroutine.

Syntax:

```
tag:                ; "Tag" name used by the GoTo Command.

GoTo tag            ; Jumps to the script statement following the "tag".

GoSub tag           ; Jumps to the script statement following the "tag".
                    ; Returns to the statement following the GoSub when a Return statement is encountered.

tag:                ; "Tag" name used by the GoSub Operation.
   :
   :
Return              ; Returns to the line after the GoSub Operation that branched to this routine.
```

Where:

tag	A marked place in the script where the **Goto** and **GoSub** Commands will branch.

Notes:

A *tag* is a marked place in the script where the **Goto** and **GoSub** commands will branch. A *tag* name must be at the start of a line, begin with a letter, contain no spaces, and end with a colon. Each *tag* must have a unique name in the script.

A **GoTo** Command branches to a specified *tag* within the script. Execution of the script will continue with the line following the *tag*. Scripts with many **GoTo** statements may be difficult to follow. Use more structured Commands like **If...Then...Else** to improve script readability.

Subroutines are sections of code that can be executed from multiple places within a script. If you find yourself repeating the same sections of code over and over within your script, this code would probably be a candidate for a subroutine. Subroutines are started by the **GoSub** Command. A **GoSub** Command branches to a specified *tag* within the script and executes statements following the *tag* until a **Return** Command is encountered. After the **Return**, execution of the script continues with the next line after the **GoSub**. Make sure that a matching **Return** is executed for every **GoSub**, or script execution will become unpredictable. **GoSub** Command can be nested 100 levels deep.

Example:

```
Again:                          ; Loops back to here

tvar = tvar + 1                 ; Count the number of loops
MsgBox  tvar

If tval < 100 Then GoTo Again   ; If we have not performed 100 loops,
                                ; then go back to top and try again.

GoSub DoItAll                   ; Next statement executed will be the MsgBox
   :                            ; statement in the DoItAll subroutine.
   :
DoItAll:
      MsgBox msg
   :
Return                          ; Return back to the statement following
                                ; the GoSub DoItAll statement.
```

Script flow Commands (Event processing)

Description: Performs various script flow control Commands based on events that occur.

Syntax:

On Error {Goto|Gosub} *tag* ; Jumps to the script statement following the "*tag*" ; when an error occurs.

Resume ; Jumps to the script statement following the ; statement that had the error that initiated the ; **ON ERROR** routine.

On HangUp {Goto|Gosub} *tag* ; Jumps to the script statement following the "*tag*" ; when a caller hangs up the phone.

On Switch{1|2|3|4} {Goto|Gosub} *tag* ; Jumps to script statement following the "*tag*" ; when a local user (or the script) changes a line ; switch.

On MainSwitch{1|2|3|4} {Goto|Gosub} *tag* ; Jumps to the script statement following the "*tag*" ; when a local user (or the script) changes a ; system switch.

OnTime *tval* ; Processes all Commands between the **OnTime**
: ; Command and the **OnTimeEnd** Command when
: ; the time specified in tval occurs.
:
OnTimeEnd

Where:

tag	A marked place in the script where the **GoTo** and **GoSub** Commands will branch.
tval	A time value that determines when **OnTime** Commands execute.

Notes:

See the section on miscellaneous system variables for related information.

The **On Error** Command allows your script to perform special routines when a script error occurs. When an error occurs, control will pass to the subroutine specified on the **On Error** Command. When the **Resume** Command is encountered in the error routine, control will pass back to the statement after the line that had the error.

The **Resume** Command is placed in the error routine to return control to the statement following the line that had the error.

The **On HangUp** Command allows your script to perform special routines when the caller hangs up the phone.

The **On Switch** Command allows your script to perform special routines when switches (1-4) for the line are turned on or off. These switches are typically used to make the script behave differently during certain conditions, as controlled by a local operator.

The **On MainSwitch** Command allows your script to perform special routines when switches (1-4) for the system are turned on or off. These switches are typically used to make the script behave differently during certain conditions, as controlled by a local operator.

Every Command between the **OnTime** and the **OnTimeEnd** Commands will be executed at the time specified on the **OnTime** Command. The “on” time is in the format of HH:MM and seconds are not allowed.

Additional notes for the **OnTime** Command:

The **OnTime** and **OnTimeEnd** Commands can only be performed in a Control Window script.

Up to 10 **OnTime** Commands can be performed in a single script.

Make sure that **OnTime** commands do not overlap. If a 1:00am **OnTime** command is executing and it takes more than an hour to complete, a 2:00am **OnTime** command would NOT execute.

Control Window scripts must start with any **OnTime** commands followed by valid Control Window Commands, followed by a **OnTimeEnd** command.

Only one (1) **OnTimeEnd** statement per **OnTime** statement allowed.

The Control Window script must complete with an **End** command before any **OnTime** commands will execute.

Put **Wait** commands in your Control Window script to keep it from "consuming" your ODT VISION system.

Example:

```
; This sample performs two sample switching functions:
;  1. When the System Switch # 1 is turned on, set ISNight to True.
;     This could be used to tell the system when the local operator
;     is no longer on duty.
;  2. When individual line switch # 1 is turned on, kill the line.
; This sample also shows how to use the On Hangup command.

On MainSwitch1 GoSub NightTime
On Switch1 GoTo EndDay
On HangUp GoTo UserTerminated
Top:
      :
If IsNight Then
      :                         ; Do this if night (no operator on duty)
Else
      :                         ; Do this if day (operator on duty)
EndIf
      :
      :
UserTerminated:                 ; Script will branch to here if the user hangs up
      :                         ;  the phone.
      GoTo Top

EndDay:                         ; Script will branch here if the operator turns on
      End                       ; the line switch #1.  This line will no longer
                                ; answer calls since the END Operation was executed.

NightTime:                      ; Script will branch here if the operator turns the
      If MainSwitch1 Then       ; system switch # 1 on or off.
            IsNight = True
      Else
            IsNight = False
      EndIf
      Return
```

```
; This example Control Window script shows how a ODT VISION system
; can be shut down between the hours of 1:00am and 3:00am with
; database updates occurring before restarting. This might be
; used when a host system would be shutting down at 1:30am for
; routine maintenance and to create a new database file required
; for the ODT VISION system to use on the next day.


; Don't forget to put WAIT statements in your script to allow the
; phone lines to get control once in a while.

; This routine wakes up at 1:00am and sees if any lines are in calls.

; Any line that is not in a call is stopped.

; The routine keeps executing until all lines have been stopped.


OnTime "1:00"

RETRY:

   linect = 1
   busy = False

LOOP:

   incall = LineStatus linect

   If incall = 2 Then
         ; If line is in a call, do nothing.
         busy = True

   Elseif incall = 1 Then
         ; If line is idle, stop the line.
         LineStop linect

   Elseif incall = 0 Then
         ; Line is not currently running....nothing to do

   Endif


   linect = linect + 1


   If linect > LineCount Then
         ; All lines have been checked.
         ; If any lines were still in a call,
         ; wait 10 seconds and try to stop them again

         If busy Then
               Wait 10
               Goto RETRY
         Endif

   Else
         ; Still more lines to check...
         ; loop up and check the next line
         Wait 1
         Goto LOOP
   Endif

OnTimeEnd
```

```
; This routine wakes up at 3:00am and does some batch database updates.
; Then is starts all of the lines.

OnTime "3:00"

   ;
   ; Do batch database updates here.
   ;
   ; Don't forget to put WAIT statements in your script to allow the
   ; phone lines to get control once in a while.
   ;

   linect = 1

LOOP2:

   LineStart linect
   Wait 10
   linect = linect + 1
   If linect <= LineCount then
      Goto LOOP2
   Endif

OnTimeEnd


; The section after all of the ONTIME Commands is used to do things
; that would only occur when the ODT VISION system is restarted.

; There does not have to be anything in this section except an END
; command which MUST be there for the ONTIME commands to start

; Don't forget to put WAIT statements in your script to allow the
; phone lines to get control once in a while.

; This area must end with an END Command.

End
```

Script flow Commands (Miscellaneous)

Description: Performs various script flow control Commands.

Syntax:

End ; Ends call processing for the current line.
Wait *seconds* ; Waits for the number of seconds specified.

Where:

seconds A numeric value with the number of seconds to pause

Notes:

The **End** Command will terminate the script for the current phone line. No other Commands will be performed for this phone line until the Operator program is restarted for this line. This Command is typically used to shut down call processing at a certain time of by the use of a switch. The **End** Command is also used in a Control Window script to start **OnTime** Command processing.

The **Wait** Command pauses the script execution for a specified number of seconds.

Example:

```
On Switch1 GoTo EndDay

Pause 10                        ; Wait 10 seconds and then continue.
      :
      :
Top:
      :
      :
      :
EndDay:                         ; Script will branch here if the operator turns on
      End                       ; the line switch #1.  This line will no longer
                                ; answer calls once the END Command has executed.
```

Flat file Commands

Description: Performs flat ASCII file maintenance.

Syntax:

Open *DOSfile* **For {Input|Output|Binary} As** *fileID* ; Open a flat ASCII file for processing.
Close *fileID* ; Close the file.
Kill *DOSfile* ; Delete a file from the disk.

Input *fileID, strvar* ; Read a string from the open file.
Output *fileID, strvar1 [, strvar2.....]* ; Write a string to an open file.

strvar = **UniqueFile** ; Returns a unique file name.

Where:

DOSfile	A string value that contains the name of a valid DOS file. See the notes section for more information.
fileID	A symbolic name constant used in the script to represent the file being processed.
strvar, strvar1, strvar2,...	String variable(s) that will be written to, or read from the file.
strvar	The result variable that will be assigned the unique file name.

Notes:

See the section on directory and flat file system variables for related information.

You must open a flat ASCII file using the **OPEN** Command before using the **Input**, **Output** or **Close** Commands. When you are done with the file, make sure the file is closed before ending the script.

There is a maximum of 128 flat files (*fileID*'s) per script. There is a system maximum of 256 flat files or Database Tables that can be opened at one time. Memory and other system limitations can cause this number to be much lower.

The *fileID* can be any name you wish the file to be known as in the script. *FileID* must be a string constant and is not case sensitive. A variable is not accepted for this parameter. The *fileID* can be surrounded by double quotes (") or brackets ([ ]) if the name contains spaces or other special characters. For example, all of the following *fileID's* are valid:

```
[Customer File]
"Customer File"
ItemFile
```

DOSFile must be a valid DOS file name with an optional relative or absolute path.

The default directory for the **Open** and **Kill** Commands are the ODT VISION Main Install Directory.

Thie **UniqueFile** Command creates a unique filename every time it is called. It is typically used to create a file name for inbound voice recordings. The filename is in the format:

lllyyyymmddhhnnss#####

where l = line #
y = year
m = month
d = day
h = hours (military)
n = minutes
s = seconds
# = unique sequential number within the call

The result of the **Input** or **UniqueFile** Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable will be converted to a string data type. If the result variable is a file variable or a read/write system variable, then the value will be converted to a type compatible with the result variable.

See the section on variables for more information on variable conversions.

Example:

```
var1 = “AA”
var2 = “BB”
var3 = “CC”
Open “FILEOUT.TXT” For Output AS MyFile    ; Create a file.
Output MyFile, var1, var2, var3            ; Write “AABBCC” to the file.
Output MyFile, “DD”, “EE”, “FF”            ; Write “DDEEFF” to the file.
Close MyFile
Open “FILEOUT.TXT” For Input As  MyFile    ; Open and Read the file created.
Input MyFile, DataIn                       ; DataIn will contain “AABBCC”.
Input MyFile, DataIn                       ; DataIn will contain “DDEEFF”.
Close MyFile                               ; Close the file
Kill “FILEOUT.TXT”                         ; Delete the file that was created.


; Get a unique file name, then give it a path and extension.
; example: "C:\RECORD\00120040112123155000 23.VOX"

filerec = UNIQUEFILE                       ; Get a unique file name.
filerec = "C:\RECORD\" & filerec           ; Add the path to the front.
filerec = filerec & ".VOX"                 ; Add the VOX extension to the end.

var = PLAY "AskRec.vox"                    ; Ask the caller to record a message.
var = RECORD filerec                       ; Record the message.
```

This page left intentionally blank

Database Table Commands

Description: Performs Database (Table) maintenance.

Syntax:

OpenDB *fileID, {index|****None***}* ; Open a DataBase Table
CloseDB *fileID* ; Close a DataBase Table

EditDB *fileID* ; Allow changes to a Table record
InsertDB *fileID* ; Insert a Table record
DeleteDB *fileID* ; Delete a Table record
UpdateDB *fileID* ; Update a Table record after an **EditDB** or **InsertDB** Command

FirstDB *fileID* ; Move to the first record in a Table
LastDB *fileID* ; Move to the last record in a Table
NextDB *fileID* ; Move to the next record in a Table
PriorDB *fileID* ; Move to the prior record in a Table

SeekDB *fileID, seekmode, strval1* [*, strval2,.....*]
; Move to the first record in a Table that meets the search criteria
; specified. Searchusing the Index specified during the **Open**.

Where:

FileID A symbolic name constant used in the script to represent the file being processed. This must be a valid Table name in the ODT VISION.MDB DataBase.

index if the Table being opened is in the ODT VISION.MDB Database Container:

This is the name of the index that was built over the Database Table.

Or

If the Table is an external (attached) table or an ODBC table:

This is the name of the Field that will be used for the SEEKDB Command.

Or

***NONE** can be specified if no index is required.

seekmode An operator that defines which records will be searched for.

Strval1, strval2, ... String values representing the key of the record(s) to look for in the Database Table.

Notes:

See the section on Database system variables for related information.

You must open a Database Table using the **OPENDB** Command before using any other Database Commands. When you are done with the Table, close it using the **CLOSEDB** Command.

There is a maximum of 128 DataBase files (*fileID's*) per script. There is a system maximum of 256 flat files or Database Tables that can be opened at one time. Memory and other system limitations can cause this number to be much lower.

FileID must be a name constant, as a variable is not acceptable for this parameter. The *fileID* must be the same name for the Table that was used in the DataBase Container. *FileID* is not case sensitive. The *fileID* can be surrounded by double quotes (") or brackets ([ ]) if the Table name contains spaces or other special characters. For example, all of the following *fileID's* are valid:

[Customer File]
"Customer File"
ItemFile

If the Table to be opened is a Table in the ODT VISION.MDB Database Container, then *Index* is the name of one of the indexes built over the Table. The index controls the order that the records will be retrieved from the Table, or it will be used for the **SeekDB** Command.

If the Table to be opened is an external (attached) Table or an ODBC Table, then *Index* is the name of the field that will be used by the SEEKDB Command. The field should be used in one of the indexes that are built over the attached or ODBC Table.

Seekmode is an operator that defines which records will be found during a **SeekDB** Command.

Seekmode can be any of the following operators.

Seekmode Operator	Test for:
<	Records Less than
>	Records Greater than
<=	Records Less than or Equal
>=	Records Greater than or Equal
=	Records Equal
<>	Records Not Equal

Example:

```
; This example opens the Customer file that is contained in the ODT VISION.MDB Container
and
; processes all of the records in the file by the name index.  The CallsToday field is
; reset back to 0 in the Customer file.
OpenDB CustomerFile, "IndexByName"
FirstDB CustomerFile            ; Move to the first record in a Table

NextRecord:
IF EOFDB Then Goto ExitRoutine
.
EditDB CustomerFile             ; Allow changes to a Table record
CustomerFile.CallsToday = 0     ; Update the call counter in the file
UpdateDB CustomerFile           ; Update a Table record after an EditDB
    .
NextDB CustomerFile             ; Move to the next record in a Table
GoTo NextRecord

ExitRoutine:
CloseDB CustomerFile            ; Close a DataBase Table



; This example opens the customer file that is contained in the ODT VISION.MDB Container
and
; asks a user for a valid customer number.  If the number is invalid, loops back and asks
; for a new number.  If the customer number is valid, add 1 to the CallsToday field and
; update the record back to the Customer file.
OpenDB CustomerFile, "IndexByCustomerNumber"
    .
GetCustomerNumber:
    .                                   ; Get customer # from the caller (CustIn)
    .
SeekDB CustomerFile, =, CustIn          ; Look for customer in the file

IF MatchDB Then                         ; If customer found......
      EditDB CustomerFile               ; Allow changes to record
      CustomerFile.CallsToday = CustomerFile.CallsToday + 1
      UpdateDB CustomerFile             ; Update record
      CloseDB CustomerFile              ; Close the Table

Else                                    ; If customer not found, tell user and
        .                               ; loop back for next customer number
      Goto GetCustomerNumber
EndIf



; This example opens the OrdersToday file (with no index) that is contained in the
; ODT VISION.MDB Container, request the customer number, item, and quantity ;and creates a
; record in the file with this information.
OpenDB OrdersToday, *NONE
    .                               ; Get customer number from user here (CustIn)
    .                               ; Get item number from user here (ItemIn)
    .                               ; Get quantity from user here (QtyIn)
InsertDB OrdersToday                ; Add a new record
OrdersToday.Date = DATE             ; update any fields in the new record
OrdersToday.Customer = CustIn
OrdersToday.Item = ItemIn
OrdersToday.Qty = QtyIn
UpdateDB OrdersToday                ; Update record
CloseDB OrdersToday                 ; Close the Table
```

This page left intentionally blank

Local User input/output and System Commands

Description: Performs interaction with the local operator or performs special functions on the ODT VISION® system.

Syntax:

Beep ; Sounds a tone that can be heard by the local operator.
PlayLocal *file* ; Plays a wave file that can be heard by the local operator.
MsgBox *prompt* [, **Beep**] ; Displays a message to the local operator

strvar = **InputBox** *seconds, prompt* [,*default*] [,**Beep**] [,**Keypad**]
; Displays an input box to get information from the local operator.

LogHigh *text* [, **Beep**] ; Always write a user entry to the line log file.
Log *text* [, **Beep**] ; Write a user entry to the line log file when logging is set to 3 or lower.
LogLow *text* [, **Beep**] ; Write a user entry to the line log file when logging is at 4 or higher.
WindowShow ; Display the line operator window for this line.
WindowHide ; Hide the line operator window for this line.
numvar = **RestartSystem** *type* ; Used in a Control Window script to shut down the system and optionally restart.
LineStart [*lineno*] ; Used in a Control Window script to start the line specified in *lineno*.
LineStop [*lineno*] ; Used in a Control Window script to stop the line specified in *lineno*.

Where:

prompt	A string value that contains a message to be displayed to the local operator.
seconds	A numeric value with the number of seconds to wait for local operator's input before accepting the default value.
default	A string value that contains the default value that is returned if the local operator does not enter a new value in the number of seconds specified, or if the *Enter* key is pressed without an entry.
text	A string value that contains the text that will be recorded into the log as a user log entry.
strvar	The result variable that will be assigned the string value that was entered by the local operator during an **InputBox** Command.
lineno	The line number to start or stop.

Notes:

The **Beep** Command produces a tone to the local operator. This is typically used to notify the operator that an event has occurred or some form of interaction is required. **Beep** is also a parameter to the **InputBox, MsgBox,** and **Log** Operations.

The **Playlocal** Command speaks a voice file (.WAV) to the local operator. This is typically used to notify the local operator that an event has occurred or some form of interaction is required.

The **MsgBox** Command is used to give a local operator an information-only message. The operator does not need to respond to the message for the script to continue processing. A **Beep** parameter can be specified if a tone should be sounded when the message is displayed.

The **InputBox** Command is used to get information from the local operator. The *prompt* is displayed to request the information required from the local operator. The *default* will be supplied if the operator does not respond within the time specified in *seconds,* or if the *Enter* key is pressed without an entry. The script will not continue until the local operator supplies the information requested or until the number of *seconds* specified has elapsed. A **Keypad** can be displayed so the user can use the mouse, instead of the keyboard, for input. A **Beep** parameter can be specified if a tone should be sounded when the request is displayed.

The **Log** (LogHigh/Log/LogLow) Commands write a user log entry to the line log with the information specified in *text*. A **Beep** parameter can be specified if a tone should be sounded when the log entry is created.

The **WindowShow** and **WindowHide** Commands are used to display or hide the Operator program window in a script. These are typically used at the start of a script to force the Operator window to display automatically.

The **RestartSystem** Command will terminate the Monitor Program and restart or shut down the ODT VISION system. This Command can only be executed in a Control Window script. The *type* parameter can be one of the following"

1 - Restart (default, if *type* is not specified)
2 - Shutdown

The **LineStart** and **LineStop** Commands starts or stops the lines specified by the *lineno* parameter. The *lineno* parameter is optional, and if it is not supplied, all lines will be started or stopped. The Line Commands can only be used in a Control Window script.

The result of the **InputBox** Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable type will be converted to a string. If the result variable is a file variable or read/write system variable, then the variable type returned will be converted to a variable type compatible with the result variable type.

See the section on variables for more information on variable conversions.

Example:

```
Beep                          ; Sound a tone

PlayLocal "starting.vox"      ; Tell local operator we are starting.

WindowShow                    ; Display the Operator window for this line.

; Request an extension from the local operator.  If no entry is made in 20 seconds,
; default to 0.  Allow entry via the on-screen keypad.
test = InputBox 20, "Enter the extension", "0", Beep, Keypad
If test = 0 Then
      MsgBox "Transferred to Operator"    ; Display a message to the operator
Else
      LogEntry = "Transferred to " CAT test     ; Write a record with the
      Log LogEntry, Beep                        ; extension to the log
EndIf
```

Phone Commands (Hook control)

Description: Performs phone line answering and hook control.

Syntax:

WaitForRing *ringcount*	Suspend script execution and wait for the phone to ring *ringcount* times.
ReWaitForRing	Return back to the **WaitForRing** statement and wait for the next call.
OffHook	Take the phone line "Off-hook". This would perform the same function as lifting the receiver to answer or place a call on a phone.
OnHook	Place the phone line "On-hook". This would perform the same function as hanging up after placing or receiving a phone call.

Where:

ringcount	A numeric value that contains the number of rings to wait for before continuing with the rest of the script.

Notes:

A *ringcount* of 0 will ignore all incoming rings which has the effect of disabling the line from accepting calls. A *ringcount* of 3 or greater is required to use Caller ID support. See the CallerID system variable for more information on Caller ID support.

The example below shows the basic flow of a phone call and how these Commands are used to control the call.

Example:

```
RingCount  =  2                ; RingCount is 3
WaitForRing  RingCount         ; Wait here until a call comes in
OffHook                        ; Take the phone off the hook
   .
   .                           ; Process the call here
   .
   .
   .
   .
OnHook                         ; Done with the call....Hang Up
ReWaitForRing                  ; Return to the WaitForRing line
```

Phone Commands (Recording)

Description: Records a message form the caller.

Syntax:

strvar = **Record** *file*[*, comp, stopchar, maxtime, maxsilence, prompt*]

Records a message to a voice file called *file* using the compression rate set by *comp*. Recording will stop after *maxtime* or *maxsilence* seconds, or if the user presses one of the *stopchar* digits. An optional tone can be generated before starting the recording.

Where:

file	A string value that contains the file name to record.
comp	A numeric value that contains the compression rate of the recording.
stopchar	A string value that contains the digits that the user can press to stop the recording.
maxtime	A numeric value that contains the maximum amount of time that a recording can take before ending automatically.
maxsilence	A numeric value that contains the maximum amount of silence that can occur before the recording will end automatically.
prompt	A true/false value that tells the system to begin the recording with a tone.
strval	The result variable that will be assigned the string value that is returned from the **Record** Operation. This value will allow you to determine what happened during the voice recording.

Notes:

File must contain a valid voice file name with an optional relative or absolute path, and an optional drive letter.

The default directory for the **Record** Command is the "Inbound Voice Files" directory. See the section on default directories for more information on how directories are specified in a script.

Comp is the compression rate that the voice recording will be made at. The default value for *comp* is 0. If an invalid number is supplied for *comp*, 0 will be substituted. The following are valid values for *comp*:

Value	Compression
0	8 bits per sample (no compression)
4	4 bits per sample
3	3 bits per sample
2	2 ²/₃ bits per sample
1	2 bits per sample

For the best sound quality in your application, use a full 8 bits per sample (value 0) for all recordings. The more bits per sample that are used in a recording, the more disk space is required to store these recordings. Most recordings should be made at a full 8 bits to be clearly heard by the person on the other end. You can use the lower bits per sample rates on "inbound" recordings to save disk space on long voice messages. "System" voice recordings are always recorded at 8 bits per sample.

Stopchar contains the digits that the user can press during the recording to terminate the recording. The default value for *stopchar* is "" (none). The special character "@" will make any digit pressed by the user a termination digit. If "@" is used, no other characters are needed in *stopchar*. Any invalid characters supplied in *stopchar* are ignored. The following are valid characters that can be in *stopchar*:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, *, #, @

Maxtime is the maximum time (in seconds) allowed for a recording. This is typically used to limit the length of an "inbound" recorded message. If a zero is supplied for *maxtime*, the maximum time cutoff is disabled. The maximum time for a recording or playback operations is 32000 seconds. If *maxtime* is not supplied, or is invalid, then 0 will be used.

Maxsilence is the maximum time (in seconds) allowed for silence during a recording. This is typically used to automatically stop a recording when the user has stopped speaking. If a zero is supplied for *maxSilence*, then the maximum silence time cutoff is disabled. The maximum value for *maxsilence* is 255 seconds. If *maxsilence* is not supplied, or is invalid, then 0 will be used.

Prompt is a true/false value that tells the system to begin the recording with a tone. The default for *prompt* is false.

The result of the **Record** Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable type will be converted to a string. If the result variable is a file variable or read/write system variable, then the data type returned will be converted to a variable type compatible with the result variable.

See the chapter on variables for more information on variable conversions.

Strval will contain a string value that is returned from the **Record** Operation. The string will contain up to 2 characters to identify why the **Record** Operation completed.

Value	Description
“” (blank)	Recording completed
(any single character)	Recording completed by the user pressing a termination key.
“MS”	Recording completed because *maxsilence* occurred.
“MT”	Recording completed because *maxtime* occurred.
“LS”	Recording completed because caller disconnected.
“DE”	Recording failed because a DOS error occurred.
“ER”	Recording failed because an error occurred.

Example:

```
; Record a message to file: MsgFile using compression rate 0.  The recording can be
; terminated by any key, a maximum silence of 10 seconds, or a maximum recording time
; of 2 minutes.  A tone will be sounded before the recording
MsgFile = “REC” + SequenceNumber
MsgFile = MsgFile + “.vox”
ChkIt = Record MsgFile, 0, “@”, 120, 10, True
If ChkIt = “DE” Then
      MsgBox “DOS Error occurred during recording”
ElseIf ChkIt = “ER” Then
      MsgBox “Error occurred during recording”
ElseIf ChkIt = “LS” Then
      GoTo EndTheCall
EndIf
```

Phone Command (Playback)

Description: Performs voice file playback to the caller.

Syntax:

Play *file* ; Puts the voice file named *file* in the "Play Queue" to be played back at a later time.
PlayRec *file*

strval = **Play** *file*[*, comp, stopchar, maxtime, prompt*]
strval = **PlayRec** *file*[*, comp, stopchar, maxtime, prompt*]
Plays back a message from a voice file called *file* using the compression rate set by *comp*. Playback will stop after *maxtime* seconds, or if the user presses one of the *stopchar* digits. An optional tone can be generated before the voice file is played.

Where:

file	A string value that contains the file name to play.
comp	A numeric value that contains the compression rate of the recording.
stopchar	A string value that contains the digits that the user can press to stop the playback.
maxtime	A numeric value that contains the maximum amount of time that a playback can take before ending automatically.
prompt	A true/false value that tells the system to begin the playback with a tone.
strval	The result variable that will be assigned the string value that is returned from the playback Command. This value will allow you to determine what happened during the voice file playback.

Notes:

There are two types of playback and speak Commands. The first type, are playback Commands that stack (the first two forms of the commands above). If no return variable is specified, "stacking" is performed, saving the playback and speak commands until all of them have been created. This allows the ODT VISION hardware to optimize the playback of the voice files for a less "choppy" speaking effect. This is typically used to speak sentences that are comprised of multiple voice files. When stacked playback Commands are executed, the voice file is added to the speaking queue and script execution continues until a playback or speak Command occurs that returns a result variable. At this point, all "stacked" voice files are played at once.

File must contain a valid file name with an optional relative or absolute path, and an optional drive letter.

The default directory for the **Play** Command is the "Outbound Voice Files" directory, and the default directory for the **PlayRec** Command is the "Inbound Voice Files" directory. See the section on default directories for more information on how directories are specified in a script.

Comp is the compression rate that a voice recording will be played back at. The default value for *comp* is 0. If an invalid number is supplied for *comp*, 0 will be substituted. The following are valid

Value	Compression
0	8 bits per sample (no compression)
4	4 bits per sample
3	3 bits per sample
2	2 ²/₃ bits per sample
1	2 bits per sample

values for *comp*:

For the best sound quality in your application, use a full 8 bits per sample (value 0) for all recordings. The more bits per sample that are used in a recording, the more disk space is required to store these recordings. Most recordings should be made at a full 8 bits to be clearly heard by the person on the other end. You can use the lower bits per sample rates on "inbound" recordings to save disk space on long voice messages. "System" voice recordings are always recorded at 8 bits per sample.
sample.

Stopchar contains the digits that the user can press during the playback to terminate the Command. The default value for *stopchar* is "" (none). The special character "@" will make any digit pressed by the user a termination digit. If "@" is used, no other characters are needed in *stopchar*. Any invalid characters supplied in *stopchar* are ignored. The following are valid characters that can be in *stopchar*:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, *, #, @

Maxtime is the maximum time (in seconds) allowed for a playback Command. This is typically not used during a playback Command. If a zero is supplied for *maxtime*, the maximum time cutoff is disabled. The maximum time for a recording or playback operations is 32000 seconds. If *maxtime* is not supplied, or is invalid, then 0 will be used.

Prompt is a true/false value that tells the system to begin the playback Command with a tone. The default for *prompt* is false.

The result of the **PlayRec** or **Play** Command can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable type will be converted to a string. If the result variable is a file variable or read/write system variable, then the variable type returned will be converted to a type compatible with the result variable.

See the chapter on variables for more information on variable conversions.

Strval will contain a string value that is returned from the **PlayRec** or **Play** Commands. The string will contain up to 2 characters to identify why the playback Command completed.

Value	Description
“” (blank)	Playback completed
(any single character)	Playback completed by the user pressing a termination key.
“MT”	Playback completed because *maxtime* occurred.
“LS”	Playback completed because caller disconnected.
“DE”	Playback failed because a DOS error occurred.
“ER”	Playback failed because an error occurred.

Example:

```
; Play back a sentence with order information.  The sample script below would say:
; "Your order amount was $127.25. Your order number is A3455. Your order date was
; July seventh, nineteen ninety-seven"

OrderAmount = 127.25
OrderNumber = "A3455"
OrderDate = "7/17/97"

; The following lines will be "stacked" until the SpeakDate Command occurs.
Play "Part1.vox"                    ; Voice file says "Your order amount was"
SpeakDollars OrderAmount
Play "Part2.vox"                    ; Voice file says "Your order number was"
Speak OrderNumber
Play "Part3.vox"                    ; Voice file says "Your order date was"

; The next line will play back all of the previous voice recordings that were "stacked"
ChkIt = SpeakDate OrderDate, "MMM DD YYYY", "@",0, False
If ChkIt = "DE" Then
      MsgBox "DOS Error occurred during playback"
ElseIf ChkIt = "ER" Then
      MsgBox "Error occurred during playback"
ElseIf ChkIt = "LS" Then
      GoTo EndTheCall
EndIf
```

Phone Commands (Speak)

Description: Performs voice playback to the caller.

Syntax:

Speak *value*
Puts the value to be spoken in the "Play Queue" to be played back at a later time.

SpeakNumbers *value*
SpeakDollars *value*
Puts the number or dollar amount to be spoken in the "Play Queue" to be played back at a later time.

SpeakDate *value*[*, format*]
SpeakTime *value*[*, format*]
Puts the date or time to be spoken in the "Play Queue" to be played back at a later time.

strvar = **Speak** *value*[*, stopchar, maxtime, prompt*]
Speaks a value to the user. Playback will stop after *maxtime* seconds, or if the user presses one of the *stopchar* digits. An optional tone can be generated before the value is spoken.

strvar = **SpeakNumbers** *value*[*, stopchar, maxtime, prompt*]
strvar = **SpeakDollars** *value*[*, stopchar, maxtime, prompt*]
Speaks the number or dollar amount in *value* to the user. Playback will stop after *maxtime* seconds, or if the user presses one of the *stopchar* digits. An optional tone can be generated before the value is spoken.

strvar = **SpeakDate** *value*[*, format, stopchar, maxtime, prompt*]
strvar = **SpeakTime** *value*[*, format, stopchar, maxtime, prompt*]
Speaks the date or time in *value* to the user. Playback will stop after *maxtime* seconds, or if the user presses one of the *stopchar* digits. An optional tone can be generated before the value is spoken.

Where:

value	A string, numeric, or date/time value that contains the information to speak.
format	A string value that contains a format that defines how a date/time value will be spoken.
stopchar	A string value that contains the digits that the user can press to stop the playback.
maxtime	A numeric value that contains the maximum amount of time that a playback can take before ending automatically.
prompt	A true/false value that tells the system to begin the playback with a tone.
strvar	The result variable that will be assigned the string value that is returned from the speak Commands. This value will allow you to determine what happened during the voice file playback.

Notes:

There are two types of playback and speak Commands. The first type, are playback Commands that stack (the first two forms of the commands above). If no return variable is specified, "stacking" is performed, saving the playback and speak commands until all of them have been created. This allows the ODT VISION hardware to optimize the playback of the voice files for a less "choppy" speaking effect. This is typically used to speak sentences that are comprised of multiple voice files. When stacked playback Commands are executed, the voice file is added to the speaking queue and script execution continues until a playback or speak Command occurs that returns a result variable. At this point, all "stacked" voice files are played at once.

The **Speak** Command will say each letter, number, or special character in *value* as it is encountered. The following characters can be used in value:

A-Z	(Letters)
0-9	(Numbers)

The **SpeakNumbers** Command will say *value* as a number, while **SpeakDollars** will say *value* as a dollar amount. The **SpeakNumbers** and **SpeakDollars** Commands have a maximum range of -999,999,999.99 to 999,999,999.99. The following are examples of dollar amounts and how they will be spoken using the **SpeakDollars** Command.

0.00	(Nothing)
0.01	One Cent
0.23	Twenty Three Cents
1.00	One Dollar
2.00	Two Dollars
1.23	One Dollar and Twenty Three Cents
22.20	Twenty Two Dollars and Twenty Cents
-22.10	Negative Twenty Two Dollars and Ten Cents

To following examples show the difference between the **Speak**, **SpeakNumbers**, and **SpeakDollars**. Using the number 123.45:

Command	Would say……
Speak	One Two Three Point Four Five
SpeakNumbers	One Hundred Twenty-Three Point Four Five
SpeakDollars	One Hundred Twenty-Three Dollars and Forty Five Cents

The **SpeakDate** and **SpeakTime** Commands are used to speak back a date or a time that is contained in *value*. The *format* string determines how the date will be recited back to the user.

See the section on formatting strings for more information on the characters that are available to use for formatting.

Stopchar contains the digits that the user can press during the playback to terminate the Command. The default value for *stopchar* is "" (none). The special character "@" will make any digit pressed by the user a termination digit. If "@" is used, no other characters are needed in *stopchar*. Any invalid characters supplied in *stopchar* are ignored. The following are valid characters that can be in *stopchar*:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, *, #, @

Maxtime is the maximum time (in seconds) allowed for a playback Command. This is typically not used during a speak Command. If a zero is supplied for *maxtime*, the maximum time cutoff is disabled. The maximum time for a recording or playback operations is 32000 seconds. If *maxtime* is not supplied, or is invalid, then 0 will be used.

Prompt is a true/false value that tells the system to begin the speak Commands with a tone. The default for *prompt* is false.

The result of speak Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable type will be converted to a string. If the result variable is a file variable or read/write system variable, then the variable returned will be converted to a type compatible with the result variable.

See the chapter on variables for more information on variable conversions.

Strvar will contain a string value that is returned from the speak Commands. The string will contain up to 2 characters to identify why the playback Command completed.

Value	Description
“” (blank)	Playback completed
(any single character)	Playback completed by the user pressing a termination key.
“MT”	Playback completed because *maxtime* occurred.
“LS”	Playback completed because caller disconnected.
“DE”	Playback failed because a DOS error occurred.
“ER”	Playback failed because an error occurred.

Example:

```
; Play back a sentence with order information.  The sample script below would say:
; “Your order amount was $127.25. Your order number is A3455. Your order date was July
; seventh, nineteen ninety-seven”

OrderAmount = 127.25
OrderNumber = “A3455”
OrderDate = “7/17/97”

; The following lines will be “stacked” until the SpeakDate Operation occurs.
Play “Part1.vox”                    ; Voice file says “Your order amount was”
SpeakDollars OrderAmount
Play “Part2.vox”                    ; Voice file says “Your order number was”
Speak OrderNumber
Play “Part3.vox”                    ; Voice file says “Your order date was”

; The next line will play back all of the previous voice recordings that were “stacked”
ChkIt = SpeakDate OrderDate, “MMM DD YYYY”, “@”,0, False
If ChkIt = “DE” Then
      MsgBox “DOS Error occurred during playback”
ElseIf ChkIt = “ER” Then
      MsgBox “Error occurred during playback”
ElseIf ChkIt = “LS” Then
      GoTo EndTheCall
```

```
EndIf
```

Phone Commands (Touch-tone Input/Output)

Description: Captures touch-tone from the caller, or sends digits to the phone line.

Syntax:

ClearDigits ; Clears the digit buffer of any previously entered digits.

strvar = **GetDigits** *digitcount, stopchar, maxtime, prompt*
Suspend script execution and wait for touch-tone digits to be entered by the caller. Assigns the digits keyed by the user to *strvar.*

strvar = **PlayGet** *file.vox, digitcount* [*, stopchar*] [*, maxtime*]
Plays the voice file (file.vox) and then suspends script execution to wait for touch-tone digits to be entered by the caller. Assigns the digits keyed by the user to *strvar.*

strvar = **PutDigits** *value*
Sends digits in *value* over the phone line as if someone keyed them on the phone keypad.

Where:

digitcount	A numeric value that contains the number of digits to wait for from the user.
stopchar	A string value that contains the digits that the user can press to stop the playback.
maxtime	A numeric value that contains the maximum amount of time that the system will wait for the digits before ending automatically.
prompt	A true/false value that tells the system to begin the request for digits with a tone.
file.vox	A string value that contains a valid voice file that will be spoken before requesting digits from the caller.
value	A string value that contains the digits to be transmitted over the phone line.
strvar	The result variable that will be assigned the string value that is returned from the speak Operations. This value will allow you to determine what happened during the voice file playback.

Notes:

See the section on Phone system variables for related information.

Stopchar contains the digits that the user can press during **GetDigits** and **PlayGet** to terminate the Command. The default value for *stopchar* is "" (none). The special character "@" will make any digit pressed by the user a termination digit. If "@" is used, no other characters are needed in *stopchar*. Any invalid characters supplied in *stopchar* are ignored. The following are valid characters that can be in *stopchar*:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, *, #, @

Maxtime is the maximum time (in seconds) that the system will wait for the requested digits. If a zero is supplied for *maxtime*, the maximum time cutoff is disabled. The maximum time for a **GetDigits** and **PlayGet** Command is 32000 seconds. If *maxtime* is not supplied, or is invalid, then 0 will be used.

Prompt is a true/false value that tells the system to begin the digit request with a tone. The default for *prompt* is false.

The **PlayGet** command was implemented in Version 6 to replace multiple lines of script code that was previously required to request digits from a caller. See the end of this section for examples of both the old and the new style of requesting digits from the caller.

The **PutDigits** command is used to perform outbound call phone processing. This allows the **ODT VISION** system to initiate phone calls and to provide or gather information from the person that answers. The **PutDigits** command can also be used to interact with a PBX system that the **ODT VISION** system is connected to.

The following table lists the valid characters that can be used in the *value* parameter when making outbound calls. Some phone boards might not support all characters or might support other characters not listed here. Refer to your voice board driver documentation for more information.

Character	Result
0-9, *, #	The corresponding touch-tone digit is dialed.
A-D	The corresponding touch-tone is dialed.
, (comma)	Dialing is paused. You can use multiple commas to provide longer pauses.
& or !	A flash hook is issued.
I	Wait for an international dial tone.
L	Wait for a local dial tone.
P	Pulse dialing is used for the digits following this character. Note that the following characters cannot be used after the P: "A B C D * #".
T	Tone (DTMF) dialing is used for the digits following this character (default).

Transferring Calls Using a PBX or Centrex

If the phoneline attached to your ODT VISION system originates from a PBX, key system, or Centrex service, you can use the **PutDigits** Command to transfer calls within the system. To do this on most phone systems, you need to generate a flash-hook signal. A flash-hook puts the phone briefly on-hook, to signal the telephone system that a service is requested. The flash-hook is generally followed by a string of touch-tones to indicate the extension.

Placing Outbound Calls

If your application places multiple calls in rapid succession, as in a telemarketing or predictive dialing application, you must make sure the phone line is placed on-hook long enough before the next call is started. Otherwise, your phone system might interpret the short period on-hook as a flash hook. The simple solution is to use the **Wait** Command with the **OnHook** Command to enable your phone system to completely hang up.

The result of the **GetDigits** or **PutDigits** Commands can be a user-defined variable, a file variable, or a read/write system variable. If the result variable is a user-defined variable, then the result variable type will be converted to a string. If the result variable is a file variable or read/write system variable, then the variable returned will be converted to a type compatible with the result variable.

See the section on variables for more information on variable conversions.

Strvar will contain a string value that is returned from the **GetDigits** Command. The string will contain up to 2 characters to identify why the **GetDigits** Command completed.

Value	Description
"" (blank)	**GetDigits** completed
(any single character)	**GetDigits** or **PlayGet** completed by the user pressing a termination key.
(multiple characters)	**PlayGet** completed and returned the user input.
"MT"	**GetDigits** or **PlayGet** completed because *maxtime* occurred.
"ML"	**GetDigits** completed because the user keyed the number of digits specified in *digitcount*.
"LS"	**GetDigits** or **PlayGet** completed because caller disconnected.
"ER"	**GetDigits** or **PlayGet** failed because an error occurred.
"DE"	**PlayGet** playback failed because a DOS error occurred.

Example:

```
ClearDigits                          ; Clear any previous digits from the buffer.
keyed = GetDigits 10, "#", 30 ; Wait for 10 new digits or until the user pressed
                                     ; the # key.  Wait a maximum or 30 seconds.
If keyed = "ER" Then
      MsgBox "Error occurred during GetDigits"
ElseIf keyed = "LS" Then
      GoTo EndTheCall
EndIf
CustomerIn = DigitBuffer


; The following shows the long code version of getting digits from the caller.

ClearDigits
Rtn = Play "EnterCust.VOX", 0, "@"
Rtn = GetDigits 10, "#", 15
Cust = DigitBuffer


; Starting with Version 6, that can be shortened to…..

Cust = PlayGet "EnterCust.VOX", 10, "#", 15

; Both versions play a voice file to the user asking for a
; customer number. The caller can enter up to 10 characters
; or terminate early by pressing the "#" key to indicate the
; end of the customer number.
; The system will wait up to 15 seconds for the user to enter the
; customer number before returning an error.
```

Phone Commands (Multi-Language support)

Description: Changes the default language files used by **ODT VISION**.

Syntax:
Language *LangTyp* [, *VoiceFilesPath*]

Where:

langtype	The language type used to speak.
voicefilespath	A string value with a valid path that contains the voice files used for the rest of the script.

Notes:

This command allows you to switch system voice files that are used for the “speaking” Commands. You can also optionally switch the path for your application voice files at the same time. This is typically used to change the language files that are used to interact with the users of ODT VISION without making any script changes.

LangTyp =	Uses .VAP file	Dialect
USAFEMALE (Default)	VVSYSTEM.VAP	AMERICAN
USAMALE	USAMALE.VAP	AMERICAN
UKFEMALE	UKFEMALE.VAP	AMERICAN
UKMALE	UKMALE.VAP	AMERICAN
SPAFEMALE	SPAFEMALE.VAP	SPANISH
SPAMALE	SPAMALE.VAP	SPANISH
CUSTOM1	CUSTOM1.VAP	AMERICAN
CUSTOM2	CUSTOM2.VAP	AMERICAN

VoiceFilesPath is full path to voice files used by your script for the language specified by the *LangType* parameter. If *VoiceFilesPath* is not specified, the default voice files path for the line is used.

Example:

```
; This sample script has already asked the caller if they want Spanish or English
; speaking prompts

If languageSelected = 1 Then
        ; User has asked for Spanish prompts
        Language SPAFEMALE, "C:\Program Files\ODT VISION\\Voice Files Spanish"
Else
        ; User has asked for English prompts
        ; this is already set so nothing has to be done
Endif

; Here is call processing.....

var = Play "AskForCust.vox"
; If Spanish, the file name "C:\Program Files\ODT VISION\\Voice Files Spanish\AskForCust.vox"
; If English, the file name "C:\Program Files\ODT VISION\\Voice Files\AskForCust.vox"
;
GoTo ENDTHECALL

ENDTHECALL:

        ; reset the prompts back to English
        Language VVSYSTEM, "C:\Program Files\ODT VISION\\Voice Files"
        ; wait for next call
        GoTo GetNextCall
```

Phone Commands (File Conversions)

Description: Performs voice file conversion.

Syntax:

VoxWav filename.VOX *;* Converts a VOX file to a WAV file.
WavVox filename.WAV *;* Converts a WAV file to a VOX file.

Where:

filename.VOX A string value that contains an existing valid VOX file name.
filename.WAV A string value that contains an existing valid WAV file name.

Notes:

The **VoxWav** and **WavVox** Commands can convert files back and forth between **VOX** (ODT VISION system) and **WAV** (PC sound card) type voice files. This is useful if a **VOX** recording created by the **ODT VISION** system needs to be heard at a standard PC or if a **WAV** file created on another PC needs to be used by the **ODT VISION** system.

Example:

```
; This example records a message from the caller and emails it.
; Get a unique file name, then give it a path and extension.
; Example: "C:\RECORD\00120040111212315500023.VOX"

filerec = UNIQUEFILE
filerec = "C:\RECORD\" & filerec
filerec2 = filerec & ".WAV"
filerec = filerec & ".VOX"

; Ask user to record a message……
;    "Please record your message at the tone....beep"
var = PLAY "AskRec.vox"
var = RECORD filerec

; Convert it to a WAV file
VOXWAV filerec

; Email it to customer service
EMAIL "custserv@ABCo.COM"
EMAILSUBJECT "ODT VISION EMail"
EMAILTEXT "From the ODT VISION System"
EMAILATTACH filerec2
EMAILSEND
```

Line Commands

Description: Returns the status of a requested phone line.

Syntax:

numvar = **LineStatus** *lineno* ;Returns the status of a specific line

Where:

Lineno	A numeric value that contains the line number to check.
numvar	The result variable that will be assigned the numeric value that is returned from the **LineStatus** Command. This value will allow you to determine the processing status of the current line.

Notes:

One of the following is returned from the **LineStatus** Command.

0 - Line is stopped. There is no Operator window running for this line.
1 - Line is running. The line is not currently in a call.
2 - Line is running. The line is currently in a call.

System variables

Date and time system variables

Description: Returns date and/or time values.

Syntax:

StartDateTime	Returns a date/time variable with the date and time at the start of the current call (When the **OffHook** command was last executed).
StartDate	Returns a date variable with the date at the start of the current call (When the **OffHook** command was last executed).
StartTime	Returns a time variable with the time at the start of the current call (When the **OffHook** command was last executed).
DateTime	Returns a date/time variable with the current date and time.
Date	Returns a date variable with the current date.
Month	Returns the current month (1-12)
Day	Returns the current day (1-31).
Year	Returns the current year (100-9999).
Weekday	Returns the current weekday number (1-7).
Time	Returns a time variable with the current time.
Hour	Returns the current hour (0-23).
Minute	Returns the current minute (0-59).
Second	Returns the current second (0-59).

Notes:
See the section on Date and Time Commands for related information.

All date/time variables are read only.

Example:

```
If Weekday = 1 Then             ; Do this section if today is a Sunday
      .
      .
      .
ElseIf Weekday = 7 Then         ; Do this section if today is a Saturday
      .
      .
      .
Else                            ; Do this section on a weekday.
      .
      .
      .
EndIf
```

Directory system variables

Description: Returns information directories used by the system.

Syntax:

CurPath	Returns the current directory for the system.
VoiceFilePath	Returns the Outbound Voice file path that was assigned during the configuration of this line.
SystemPath	Returns the System Voice file path that was assigned during the configuration of this line.

Notes:
See the section on Flat file Operations for related information.

All directory variables except **CurDir** are read only. Changing **CurDir** will change the current directory for ALL lines. **CurDir** must be assigned a valid drive and directory or an error will occur.

Example:

```
TestFile = InboundDir Dcat "Record.vox"
If Exists TestFile Then                    ; A recording named "Record.vox"
      .                                    ; exists in the Inbound Voice files
      .                                    ; directory.
      .
EndIf
```

Flat file system variables

Description: Returns information about the status of Flat files.

Syntax:

LOF Returns the number of characters in the last Flat file accessed.

EOF Returns **TRUE** (-1) if the last Flat file accessed is at the end of the file, otherwise, it returns **FALSE** (0).

Notes:

See the section on Flat file Operations for related information.

All Flat file variables are read only.

Example:

```
Open "Customer.Dat" For Input As CustomerFile
FileSize = LOF                                  ; Determine the file size
GetNextRecord:
If EOF Then GoTo EndOfFile                      ; End of file yet?
Input DataIn                                    ; Get the next record
      .
      .                                         ; Process the record here
      .
GoTo GetNextRecord                              ; Get another record
```

Database file system variables

Description: Returns the status of the last Database file accessed.

Syntax:

LOFDB Returns the number of records in the Database file.

BOFDB Returns **TRUE** (-1) if the last Database file accessed is at the beginning of the file, otherwise, it returns **FALSE** (0).

EOFDB Returns **TRUE** (-1) if the last Database file accessed is at the end of the file, otherwise, it returns **FALSE** (0).

MatchDB Returns **TRUE** (-1) if the last "Seek" on a DataBase file found a match, otherwise, it returns **FALSE** (0).

Notes:

See the section on Database Commands for related information.

All Database file variables are read only.

Example:

```
; This sample opens the Customer file that is contained in the ODT VISION.MDB Container
and
; processes all of the records in the file by the name index.  The CallsToday field is
; reset back to 0 in the Customer file.
OpenDB CustomerFile, "IndexByName"
FirstDB CustomerFile           ; Move to the first record in a Table
NextRecord:
IF EOFDB Then Goto ExitRoutine
.
EditDB CustomerFile            ; Allow changes to a Table record
CustomerFile.CallsToday = 0
UpdateDB CustomerFile          ; Update a Table record after an EditDB
    .
NextDB CustomerFile            ; Move to the next record in a Table
GoTo NextRecord

ExitRoutine:
CloseDB CustomerFile           ; Close a DataBase Table


; This sample opens the customer file that is contained in the ODT VISION.MDB Container
and
; asks a user for a valid customer number.  If the number is invalid, looks back and asks
; for a new number.  If the customer number is valid, add 1 to the CallsToday field and
; update the record back to the Customer file.
OpenDB CustomerFile, "IndexByCustomerNumber"
    .
GetCustomerNumber:
    .                                  ; Get customer number from user here (CustIn)
    .
SeekDB CustomerFile,=, CustIn          ; Look for customer in file
IF MatchDB Then                        ; If customer found......
      EditDB CustomerFile              ; Allow changes to record
      CustomerFile.CallsToday = CustomerFile.CallsToday + 1
      UpdateDB CustomerFile            ; Update record
      CloseDB CustomerFile             ; Close the Table
Else                                   ; If customer not found, tell user and
         .                             ; look back for next customer number
      Goto GetCustomerNumber
```

```
EndIf
```

Phone system variables

Description: Returns values of phone related Commands or call processing.

Syntax:

DigitBuffer Returns the digits entered by the user during the last **GetDigits** Command.

CallerID Returns the Caller ID information received from the Phone Company or PBX.

Notes:

See the section on Phone Commands (Digit Input/Output) for related information.

The **DigitBuffer** and **CallerID** system variables are read only.

The **CallerID** Command is not supported on all phone boards and PBX systems.

The **CallerID** Command will return one of the following results:

OUT OF AREA
UNAVAILABLE
BLOCKED
{Date} {Time} {Caller's phone #} {Caller's name - if provided}

An attempt to retrieve Caller ID information from a board or PBX that does not support Caller ID returns “UNAVAILABLE”.

The **WaitForRing** command must be set to at least two (2) rings, since the Caller ID information is transmitted between the first and second ring.

Troubleshooting Caller ID support

Possible reasons why the CallerID Command would return “UNAVAILABLE”.

- The phone hardware you are using does not support Caller ID.
- You are not using the correct version of the drivers.
- The phone line connected to the phone board does not provide Caller ID information.
- The PBX you are attached to does not pass Caller ID information
- The **WaitForRing** Command is not waiting for at least 2 rings.

Example:

```
; Wait for the call

WaitForRing 3

CallerInfo = CallerID

; Request the customer number form the caller

Rtn = Play "getCustNo.VOX"

ClearDigits                     ; Clear any previous digits from the buffer.
keyed = GetDigits 10, "#", 30 ; Wait for 10 new digits or until the user pressed
                                ; the # key.  Wait a maximum or 30 seconds.
If keyed = "ER" Then
      MsgBox "Error occurred during GetDigits"
ElseIf keyed = "LS" Then
      GoTo EndTheCall
EndIf
CustomerIn = DigitBuffer
```

Miscellaneous system variables

Description: Returns or sets values in the ODT VISION® system.

Syntax:

True	Returns -1.
False	Returns 0.
Yes	Returns -1.
No	Returns 0.
ErrorNo	Returns the Error number of the last error encountered.
SeqNo	Returns the sequence number of the call for all lines.
MainSwitch 1 **MainSwitch 2** **MainSwitch 3** **MainSwitch 4**	Returns **TRUE** (-1) if the switch for the system is **ON**, otherwise, returns **FALSE** (0).
LineNo	Returns the Line number that this script is running under.
LineSeqNo	Returns the sequence number of the call for this line only.
LineCount	Returns the total number of lines that CAN be started on this system.
Switch 1 **Switch 2** **Switch 3** **Switch 4**	Returns **TRUE** (-1) if the switch for the line is **ON**, otherwise, returns **FALSE** (0).

Notes:

See the section on Script Flow Operations (Event Processing) Operations for related information.

All miscellaneous variables are read only except **ErrorNo** and **MainSwitch(x)** and **Switch(x)**.

The **LineCount** system variable contains the total number of lines that were configured in the properties window of the Monitor program. This is not the number of lines currently running on the system, but the maximum number of lines that could be running.

MainSwitch variables are used to globally control the way a script works. These are typically used to control an end-of-day or operator on-duty/off-duty condition in the script. **Switch** variables are similar, except they are used to control only the line that the switch was set for. **Main** and **Line Switches** are set to off (0) at the start of the script. If **Switch** variables are changed, the physical switch on the screen will be changed and any applicable **On Switch** routines will be executed. Assigning a **Switch** variable a 0 value will turn the switch off, any other value will turn the switch on.

Example:

```
On Switch1 GoSub EndDay
       :
       :
Top:
       :
       :
       :
       :
EndDay:                              ; Script will branch here if the operator turns on
If Switch1 = True Then               ; the line switch #1.  This line will no longer
       End                           ; answer calls since the END operation was executed.
Else
       Resume
EndIf


; Create a name for a recording file that is comprised of the line number and sequence
; number of the call.
FileName = Format LineNo,"##"
FileName = FileName + LineSeqNo
FileName = FileName + ".vox"
```

Formatting

Formatting numbers

To format numbers, you can use the predefined format names or create user-defined formats using characters that have special meaning when used in a format string. See the following pages for information on the predefined formats, or the allowable characters in a user-defined format.

If format is blank and value is numeric, then **Format** will return a string formatted like the **String** Operation. Positive numbers converted to strings using the **Format** Operation will not have the leading space reserved for the sign.

Unless the format argument contains one of the predefined formats, a format expression for numbers can have from one to three sections separated by semicolons.

If you use	The result is
One section only	The format expression applies to all values.
Two sections	The first section applies to positive values and zeros, the second to negative values.
Three sections	The first section applies to positive values, the second to negative values, and the third to zeros.

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

```
"$#,##0;($#,##0)"
```

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

```
"$#,##0;;\Z\e\r\o"
```

Some sample format expressions for numbers are shown below. (These examples all assume that Country is set to United States in the International section of the Control Panel.) The first column contains the format strings. The other columns contain the output that results if the formatted data has the value given in the column headings.

Format (fmt)	Positive 5	Negative 5	Decimal .5l
Null string	5	-5	0.5
0	5	-5	1
0.00	5.00	-5.00	0.50
#,##0	5	-5	1
#,##0.00;;	5.00	-5.00	0.50
$#,##0;($#,##0)	$5	($5)	$1
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50
0%	500%	-500%	50%
0.00%	500.00%	-500.00%	50.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01
0.00E-00	5.00E00	-5.00E00	5.00E-01

Formatting dates and times

You can format date and time values by using date and time formats or number formats because date/time values are stored as real numbers. To format dates and times, you can use either the commonly used formats that have been predefined or create user-defined formats using standard characters that have special meaning when used in a *format* parameter.

The following are examples of user-defined date and time formats:

Format	Display
m/d/yy	12/7/58
d-mmmm-yy	7-December-58
d-mmmm	7 December
mmmm-yy	December 58
hh:nn AM/PM	08:50 PM
h:nn:ss a/p	8:50:35 p
h:nn	20:50
h:nn:ss	20:50:35
m/d/yy h:nn	12/7/58 20:50

Formatting Strings

Strings can also be formatted with the **Format** Command.

Predefined formats for numeric values

The following table displays the predefined formats that can be used for numbers:

Format Name	Description
General Number	Display the number with no thousand separators.
Currency	Display number with thousand separator. Display negative numbers enclosed in parentheses Display two digits to the right of the decimal separator.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator. Display two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (%) appended to the end Display two digits to the right of the decimal separator.
Scientific	Display number in standard scientific notation.
Yes/No	Display No if number is 0. Display Yes if number is positive or negative.
True/False	Display False if number is 0. Display True if the number is positive or negative.
On/Off	Display Off if number is 0. Display On if the number is positive or negative.

User-defined formats for numeric values

The following table displays the characters you can use to create a user-defined format string for numbers:

Character		Function
(none)		Display the number with no formatting.
0	Digit placeholder.	Display a digit or a zero. If there is a digit in the number being formatted in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.
#	Digit placeholder.	Display a digit or nothing. If there is a digit in the expression being formatted in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros are not displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
.	Decimal placeholder.	The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. If you want a leading zero to always be displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator instead. The actual character used as a decimal placeholder in the formatted output depends on the Number Format specified in the International section of the Windows Control Panel. For some countries, a comma is used as the decimal separator.
%	Percent placeholder.	The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.

, Thousand separator.	The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a comma surrounded by digit placeholders (0 or #). Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." You can scale large numbers using this technique. For example, you can use the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent commas in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format specified in the International section of the Control Panel. For some countries, a period is used as the thousand separator.
- + $ ( ) space Display a literal character.	To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
\ Display the next character in the format string.	Many characters in the format expression have a special meaning and cannot be displayed as literal characters unless they are preceded by a backslash. The backslash itself is not displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\). Examples of characters that cannot be displayed as literal characters are: the date- and time-formatting characters: (a, c, d, h, m, n, p, q, s, t, w, y, and /:), the numeric-formatting characters (#, 0, %, E, e, comma, and period): and the string-formatting characters (@, &, <, >, and !).
"ABC" Display the string inside the double quotation marks.	To include a string in fmt from within your script, you must use **Char** 34 Operation to enclose the text (34 is the ANSI code for a double quotation mark).

Predefined formats for date/time values

The following table displays the predefined formats that can be used for dates and times:

Format Name	Description
General Date	Display a date and/or time. For real numbers, display a date and time. (e.g. 4/3/93 05:34 PM) If there is no fractional part, display only a date (e.g. 4/3/93) If there is no integer part, display time only (e.g. 05:34 PM).
Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium Date	Display a date in the same form as the Short Date, as defined in the International section of the Control Panel, except spell out the month abbreviation.
Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control Panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format (e.g. 17:45)

This page left intentionally blank

User-defined formats for date/time values

The following table displays the characters you can use to create a user-defined format string for dates and times:

Character	Function
: Time separator.	The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator depends on the Time Format specified in the International section of the Control Panel.
/ Date separator.	The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in the formatted output depends on Date Format specified in the International section of the Control Panel.
c	Display the date as ddddd and display the time as ttttt, in that order. Only date information is displayed if there is no fractional part to the date serial number; only time information is displayed if there is no integer portion.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday-Saturday).
ddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Short Date setting in the International section of the Windows Control Panel. The default Short Date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Long Date setting in the International section of the Control Panel. The default Long Date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday.)
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	Display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the year as a two-digit number (00-99).
yyyy	Display the year as a four-digit number (100-9999).

h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
t t t t t	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 PM.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 PM.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 PM.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 PM.
AMPM	Use the 12-hour clock and display the contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM.

User-defined formats for string values

The following table displays the characters you can use to create a user-defined format string for strings:

Character		Meaning
@	Character placeholder.	Display a character or a space. If there is a character in the string being formatted in the position where the @ appears in the format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
&	Character placeholder.	Display a character or nothing. If there is a character in the string being formatted in the position where the & appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an ! character in the format string. See below.
<		Force lowercase. All characters are displayed in lowercase format.
>		Force uppercase. All characters are displayed in uppercase format.
!		Force placeholders to fill from left to right instead of right to left.

Compiler limitations

Maximum script Lines	32000
Maximum Constants in script	5000*
Maximum User variables in script	1000*
Maximum File variables in script	1000*
Maximum Tags in script	5000*
Maximum nesting levels in script	4000*
Maximum nesting levels in script during execution	100
Maximum Flat files in script	128
Maximum Database Files in script	128
Maximum Files opened at one time for all scripts	256

* NOTE
These values are the defaults that were set when the system was installed. Any value marked with an asterisk (*) can be changed by modifying the Compiler section in the **ODT VISION.INI** file located in your **ODT VISION®** system directory.

All of these values may be limited by other factors in your system, such as available memory, the speed of your system, or limitations of the operating system.

Error summary

The following is a list of errors that can be displayed by the Compiler program:

00003 Return without GoSub
00020 Resume without error
00051 Internal error
00054 Bad file mode
00064 Bad file name
00067 Too many files
00639 Field type is illegal
02431 Syntax error
02448 Can't set value.

09000 Invalid Operation.
09001 Line could not be parsed.
09002 Double quote (") found inside of File Variable brackets ([ ]).
09003 Left bracket ([) missing
09004 Right bracket (]) missing
09005 Semicolon (;) found inside of File Variable brackets ([ ]).
09006 Missing double quote (").
09007 Invalid character (*char*) in Field Name (*fieldname*).
09008 Variable (variable) cannot be modified.
09009 Tag not found (*tagame*).
09010 Undefined File Name (*filename*) for a File Variable (*filevariable*).
09011 Left parentheses ‘(‘ missing
09012 Right parentheses ‘)‘ missing

09099 Too few parameters.
09100 Too many parameters.
09101 Too many Constants in script.
09102 Too many File Variables in script.
09103 Too many Variables in script,
09104 Too many Tags or IF's in script.
09105 Too many Nesting Levels (IF's).
09106 Too many Flat Files.
09107 Too many Database Files.
09110 Bad File name
09111 Bad Database File name
09112 Bad Variable name
09113 Bad File Variable name
09114 Bad Tag name

09210 Switch Variable not 1-4.
09211 Line Switch Variable not 1-4.

09301 Comparison operator not =, <, >, <=, =< , >=, =>, >< or <>.
09302 Duplicate Tag Name (*tagname*).

09401 ENDIF without IF

09999 Internal Error

The following is a list of some of the errors that can be displayed by the Operator program:

(General Errors)
00003 Return without GoSub
00020 Resume without error
00051 Internal error
00054 Bad file mode
00064 Bad file name
00067 Too many files
00639 Field type is illegal
02431 Syntax error
02448 Can't set value.

(Errors 10000-19999 are program start errors)

10000	TTITIF.DLL not available	(Hard Error)
10xxx	Error on TIFSUPINI	(Hard Error where xxx is an SI code)
11xxx	STOPPWRLINE Error	(Where xxx is an AI code)
12xxx	STARTPWRLINE Error	(Where xxx is an AI code)
17xxx	Error on TIFLINGET	(Where xxx is a line number)
18xxx	Error on TIFSUPTRM	(Where xxx is a line number)

(Errors 30000-39999 are line start errors)
30000 Script not found: *script*
30001 Script Damaged *script*

(Errors 40000-69999 are script execution errors.....the script name, compiled line number, and opcodes are returned.)
40xxx SPEAKER Error
41xxx SETLEVEL Error
42xxx MICROPHONE Error

50xxx CLEARDIGITS Eror
51xxx LINEENABLE Error
52xxx SETDIGITRECOG Error
53xxx SETRELAYS Error
54xxx STOPLINE Error
55xxx PLAYINDEX Error
56xxx RECORDAUDIO Error

69003 Return without GoSub
69020 Resume without error
69054 Bad file mode
69064 Bad file name
69067 Too many files
69448 Can't set value.
69639 Field type is illegal

(Errors 70000-89999 are event errors.....the script name, compiled line number, opcode, event, and event data are returned.)
70xxx Unexpected Event (*event*) still in Queue
171xxx Unexpected Event (*event*) occured during Opcode: *opcode*
72000 Busy count exceeded during OpCode: *opcode*

90000 - 99999
(Errors 90000-99999 are DOS errors)
90xxx DOS error during phone operation

The following tables list SI and AI sub-codes returned on an error:

(SI codes)

1	Cannot INIT DOS Extender
2	DOS TSR not active
3	Invalid Line/Port number
64	Insufficient Memory
66	Controller file not found
67	Controller path not found
69	Attempt to link to a task
70	Library requires separate segments
74	Incorrect Windows version
75	Invalid EXE
76	OS/2 application
77	DOS 4.0 application
78	Unknown EXE type
79	Protected Mode wrong version
80	Attempt to load second instruction
81	Large Frame EMS error
82	Attempt to load Protected Mode
97	Error setting PSP
98	Error setting window handle

(AI codes)

2	No Digits in Queue
3	System Active
4	System Not Active
6	Invalid Function in AH
8	Invalid Parameter
9	Invalid Line Number
10	Line is Busy
15	No Terminating Option Specified
142	Applications Interface Busy
143	TSR Not Installed

Sample vvoice32.ini file

```
Visual Voice]
PhoneLine=1
FileFormat=3
MaxCallRings=6
InterDigitStart=1
VisualVoicePath=C:\Program Files\VisualVoicePro5
SystemFilesPath=C:\WINDOWS\Desktop\ODT VISION\System Voice Files
ServerMode=0
ConserveThreads=1
Aculab=0
AutoAllocateLine=0
Blocking=0
ConvertFileFormats=1
PhoneLineType=0
MaxRecordTimeUnits=0
MaxSilenceUnits=0
MaxNonSilenceUnits=0
ExtendedDialString=0
EventWaitTime=1000

[Test Mode]
TestMode=on
IndexedFileExtension=WAP
IndexedFileFormat=8
FileExtension=WAV

[Channel Parameter Block]
intflg=7
'Return vvpNoAnswer after 15 seconds
noanswer=3000

[Control Block]
flashtm=50
pausetm=200
```

```
[Tone Definitions]


; Local dial tone
TID_DIAL_LCL=400,125

; International dial tone
TID_DIAL_INTL=400,125

; Special ("extra") dial tone
TID_DIAL_XTRA=400,125

; Busy signal (detected as a single tone)
TID_BUSY1=500,200,0,0,55,40,55,40,4

; Ringback (detected as a single tone)
TID_RNGBK1=438,138,0,0,130,105,580,415,0

; Busy signal (detected as a dual tone)
TID_BUSY2=500,200,525,175,55,40,55,40,4

; Ringback (detected as a dual tone)
TID_RNGBK2=438,138,438,138,130,105,580,415,0

; Fax or modem tone (receive)
TID_FAX1=2150,150,0,0,25,25,0,0,0

; Alternate fax or modem tone (send)
TID_FAX2=1100,50,0,0,25,25,0,0,0
```

```
; Describe any custom tones (not touch tones) you want to detect here.

; Custom tone definition for detecting a fax machine 'send' signal.
Creates a
; single tone cadence pattern which listens for a 1100Hz tone (20 Hz
; deviation).  The cadence is on for 1/2 second (0.1 second deviation),
; followed by 3 seconds of silence (0.1 second deviation).  This pattern
; must occur 2 times before the ToneDetected event is triggered.
T1=1100, 20, 50, 10, 300, 10, 2

; Custom tone definition for detecting a fax machine 'receive' signal.
Creates a
; single tone of 2085 Hz (20 Hz deviation).  The event is triggered when
; the leading edge (the beginning) of the tone is heard.  This tone
should
; be similar to a signal from a modem as well.
T2=2085, 20, L

; Custom tone definition for detecting a standard North America
dialtone.
; Creates a single tone of 400 Hz (25 Hz deviation).
T3=400,25,50,-50,0,0,0

; Custom tone definition for detecting a standard North America busy
signal.
; Creates a dual tone of 480 Hz (30 Hz deviation) and 620 Hz (40 Hz
deviation).
; The cadance is on for 1/2 second (0.1 second deviation), followed by
1/2
; seconds of silence (0.1 second deviation). This pattern must occur 2
times
; before the ToneDetected event is triggered.
T4=480,30,620,40,50,5,50,5,2

; Custom tone definition for detecting a standard North America fast
busy
; signal. Creates a dual tone of 480 Hz (30 Hz deviation) and 620 Hz
; (40 Hz deviation). The cadance is on for 1/4 second (0.1 second
deviation),
; followed by 1/4 seconds of silence (0.1 second deviation). This
pattern
; must occur 2 times before the ToneDetected event is triggered.
T5=480,30,620,40,25,5,25,5,2

[Tone Presentation]
; Describe how you want custom tones presented here.

[Prompt Manager]
LongFileNames=0
RecordGain=1

[Voice Service]
AutoShutDown=1
EnableTaskbarIcon=0
LogFile=CALL.LOG
LogFileSize=100
LogFilePrefix=CALLS
LogFileType=0
```

This page left intentionally blank

Index

This page left intentionally blank